LINPROG - A Linear Programming Solver. Mathematical description and model applications

Kirkegaard, P.; Grohnheit, Poul Erik

Publication date:
1995

Document Version
Publisher's PDF, also known as Version of record

RISØ

Risø-R-707(EN)

LINPROG - A Linear Programming Solver

Mathematical description and model applications

Peter Kirkegaard and Poul Erik Grohnheit

Risø National Laboratory, Roskilde, Denmark
June 1995

Abstract A FORTRAN 77 computer code LINPROG has been developed at Risø for solving medium- to large-scale linear programming problems. It runs on a wide range of computer platforms, including PC's. LINPROG uses the revised simplex method with the Forrest-Tomlin updating scheme of the inverse basis. Sparse-matrix techniques are applied throughout. An efficient presolve module is incorporated. A comprehensive test and verification study has been performed with data sets provided by local users and in the literature. The computer environment for different platforms are described as well as the design of models that is needed for generating large linear programming problems. Examples of model studies using LINPROG as a solver are given, focusing on solutions of optimization models for national energy systems that are described as a network of energy flows, where discounted costs over 20-30 years are minimized. Solution of optimization models for national energy systems and for energy planning is a main application field of LINPROG.

ISBN 87–550–1926-9
ISSN 0106–2840

Grafisk Service · Risø · 1995

Contents

Preface

This document is a major revision of the now obsolete report [1], which described the 1990 version of the linear programming solver LINPROG. Since then, the code has been extended and improved in various ways, maintaining the backwards compatibility in input specification. The most important new facility is the socalled presolve or reduction module. The solver has been made portable to most widely used computer platforms including PCs. Moreover we have added a discussion of important application fields, mainly in energy-environment planning and economics.

The authors wish to thank Antti Lehtilä and Pekka Pirilä from VTT Energy in Finland. They installed and tested LINPROG at VTT under OS/2 and suggested various improvements in the LINPROG coding. In particular they devised a modification of the inversion procedure based on linked lists, which resulted in a significant reduction of computing time for large problems.

June 1995
Peter Kirkegaard, Poul Erik Grohnheit

1 Introduction

LINPROG is a solver for linear programming (LP) problems. It is a "stand-alone" code written in standard-conforming ANSI FORTRAN 77, except for a few small subroutines (see Appendix B). However, its use is not restricted to a FORTRAN environment. All data transfers between LINPROG and the user are made via files. The input matrix format complies with the de-facto MPS standard for commercial LP codes.

One of our main design criteria for LINPROG was to keep the program manageable in size. We have therefore concentrated on implementing what we consider to be the most important facilities, abandoning many of the more sophisticated capabilities found in larger commercial systems.

The code has undergone a great deal of development, enhancement and modifications over the years to keep abreast with the demands from the users of solving bigger and bigger problems efficiently. The current version can solve LP problems with thousands of variables and constraints, using sparse-matrix techniques. It permits matrices up to about 16 000 × 16 000 with well above 100 000 nonzero elements.

LINPROG is capable of solving general LP problems, but is particularly efficient for certain important problem classes. For example, a widely used application of linear programming is cost minimization under a set of linear constraints for materials and cash flows in a system with capacity limitations. Such applications of linear programming techniques for energy systems analysis range from the physical or chemical processes within power stations or refineries in subseconds timescale to the development of international energy infrastructure over decades.

The current version of LINPROG is based on the *simplex method* (see Section 4.1). In recent years much progress is reported on the use of so-called *interior-point methods* .. These methods compare favorable with simplex for very large problems. We believe that most problems in energy system analysis, which is our primary target area, can still be solved just as efficiently by the simplex method. One reason is the great reduction power of the new presolve module in LINPROG for this type of problems. However, future developments may give rise to a reconsideration of this point of view.

There are several reasons why we have developed our own LP system at Risø. Historically this approach is linked to a tradition with close interaction between developers and users of optimization software. With our own code it is easier to provide programmatic interfaces for the application software and to ensure the portability of the code.

LINPROG is able to run on many different computer platforms. The range of computers, where LINPROG has already been made to run, covers DEC (VAX, Alpha, Ultrix), the UNISYS A-series, Apollo DN10000, HP9000, IBM/R6000, SUN 4/75, Silicon Graphics, Convex 220, and PCs using 386 and 486 based FORTRAN compilers running under DOS (FTN77/386 from Salford University, Lahey F77L-EM/32 and LF90) or OS/2 (WATCOM) (Section 5.1).

Much effort has been invested in testing LINPROG. In addition to LP test problems supplied by Risø researchers, we have collected a great deal of sample cases from outside. The NETLIB collection of LP data sets, which is in the public domain [2], is the most important source and has proved very valuable in debugging LINPROG.

Risø is now making an effort to commercialize LINPROG, and it has been sold to a few users of the program complex EFOM, which is used for optimization of large energy systems (Chapter 8). LINPROG is available from Risø National Laboratory, either as a stand-alone tool or included in the EFOM package. Demonstra-

tion copies of both packages may be available for potential new users as "invited shareware".

Among other things, it is the intention of this report to bring the experience from mathematical tools that was gained during the old age of mainframes, punched cards and lineprinters into the modern world of graphical user interfaces. Many of those modelling tools that for years ago required a designated hardware and a computer organization can now be run at a standard general-purpose office computer.

The contents of this report is organized in such a way that Chapter 3, the LINPROG User's Guide, contains all the practical information a potential user will need in order to prepare an LP input for LINPROG, solve the problem, and interpret the results. The following documentation of the mathematical methods and programmatic details in Chapter 4 is not needed for routine running of LINPROG. The remaining part of the report deals with computer platforms for LINPROG and describes the testing of LINPROG; moreover it explains what is meant by a "model environment" for LINPROG, and finally there is a thorough discussion of applications in energy production and energy system analysis.

2 What is an LP problem?

The computer code LINPROG is intended for solving medium- and large-scale problems of *linear programming* (LP) in which we want to minimize (or maximize) a given linear function, called the *objective function*, subject to a set of linear equations and inequalities, called *constraints* (the equivalent terms *restraints* and *restrictions* are also in common use.) Let us write our *linear program* in the following form:

$$\left.\begin{array}{lll} \text{Minimize} & z = \sum_{j=1}^{n_s} c_j x_j & \\ \text{subject to} & \sum_{j=1}^{n_s} a_{ij} x_j \quad \mathcal{R}_i \quad b_i \ (i = 1, \ldots, L) & \\ \text{and} & x_j \geq 0 \ (j = 1, \ldots, n_s). & \end{array}\right\} \qquad (1)$$

The symbol $\mathcal{R}_i$ stands for a *restriction type*, which must be either $\leq$, $\geq$, $=$, or dummy (meaning there is no restriction). The problem (1) has n_s nonnegative *structural variables* and L constraints.

Linear programming as a mathematical method is part of a broader field called operations research or mathematical programming. Historically, the development of LP techniques was linked to the advent of high-speed computers around 1950, and early applications included military planning in the post World War II era. Nowadays linear programming methods are being used in many application fields. Typically the LP models are concerned with the allocation of scarce resources among a number of products or activites. Particularly common are production and transport problems, which play a significant role in industry.

At Risø LINPROG and its predecessors have been used to solve linear programs for finding optimal fuel management schemes in a nuclear reactor, for predicting flow distributions of various types of energy in complex nationwide energy systems, and for allocating scarce resources in economic planning. These applications will be discussed at length in Chapter 8.

Though energy-economic planning is a main application field at Risø, LP has a much broader general application range. Bland [3] gives a semipopular survey of linear programming from which we quote a number of typical applications. His leading example is a brewery producing ale or beer from the raw materials corn, hops and barley malt. Given the recipes for the two products and their sales prices, together with the limitations on raw materials, then how can the brewer devise a production plan that optimizes his profit? This is an example of a so-called blending problem. In particular the petroleum industry solves such blending problems, and they also use LP models for controlling refinery operations, including catalytic cracking, destillation and polymerization. Many other industries take advantage of LP. Prominent examples are paper products, food distribution, agriculture, steel and metalworking.

To give some further examples we could mention the use of LP in forest planning, and, more recently, its use in image restoration models. Moreover the optimal design of truss structures may also be formulated as an LP problem in which one minimizes the weight of the construction. In numerical mathematics LP is sometimes used as a tool for solving approximation problems, for example when piecewise linear functions are being fitted to data.

There are certain important problem classes that in principle could be solved by LP but nevertheless fall outside the scope of a general-purpose solver as LINPROG. This is because there exist much more efficient special-purpose methods,

often based on optimization on networks, for these tasks. One such class is the so-called assignment problem, where m jobs are assigned to n workers. Examples are airline fleet assignments and other staff-scheduling problems.

We give below an illustrative example of a small LP problem and choose for this purpose the classical *diet problem.* Suppose for example that the budget manager of a nursing home wishes to purchase, at minimum cost, suitable quantities of a number of available food types, so that the daily diet provides the occupants with prescribed minimum values of nutrients (protein, vitamin, etc.). In our example, which is taken from [4], there are $n_s = 3$ food types: poultry, spinach, and potatoes. The daily requirements of nutrients per person are at least 65 grams of protein, 90 grams of carbohydrate ("energy"), 200 milligrams of calcium, 10 milligrams of iron, and 5000 international units of vitamin A. Table 1 shows the costs and nutritive food values per gram of each food type. If the quantities of poultry,

Table 1. Nutritive value of foods (from Nazareth)

	Poultry	Spinach	Potatoes
Costs (cents)	0.40	0.15	0.10
Protein (g)	0.20	0.03	0.02
Carbohydrate (g)	0	0.03	0.18
Calcium (mg)	0.08	0.83	0.07
Iron (mg)	0.014	0.02	0.006
Vitamin A (I.U.)	0.80	73	0

spinach, and potatoes, are denoted x_1, x_2, x_3, respectively, we can formulate the diet problem as a linear program conforming with (1):

$$\left.\begin{array}{lll} \text{Minimize} & z = & 0.40x_1 + 0.15x_2 + 0.10x_3 \\ \\ \text{subject to} & & 0.20x_1 + 0.03x_2 + 0.02x_3 \geq 65 \\ & & \qquad 0.03x_2 + 0.18x_3 \geq 90 \\ & & 0.08x_1 + 0.83x_2 + 0.07x_3 \geq 200 \\ & & 0.014x_1 + 0.02x_2 + 0.006x_3 \geq 10 \\ & & 0.80x_1 + \quad 73x_2 \qquad\quad \geq 5000 \\ \text{and} & & x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0. \end{array}\right\} \qquad (2)$$

We shall return to this example in connection with the User's Guide for LINPROG, Chapter 3.

The formulation (1) may be considered as the default problem setup for LINPROG, for as we shall see later, the code can deal with a number of modifications and extensions to (1). For example, it can be used to maximize instead of minimize. Moreover the nonnegativity conditions $x_j \geq 0$ for structural variables can be replaced by two-sided bounds $\ell_j \leq x_j \leq u_j$, or x_j may be either a fixed or a free variable. Other common LP extensions, such as parametric and integer programming, are not included in LINPROG.

3 LINPROG User's Guide

In modern usage of LP software the solver itself will often be embedded in a high level modeling system. By expressing the problem in such a "model language", the user is relieved from dealing with the input/output interface of the solver. For compatibility reasons this interface design follows old traditions, which today may be felt as rather awkward and rigid, and here LINPROG is no exception.

But if you want to use LINPROG as a stand-alone tool, or you plan to build it into a larger modeling framework, you will need a precise specification of all its input/output communications. Therefore, at this place we shall present a User's Guide for LINPROG, and in doing so we make repeated use of the small sample problem in the previous chapter.

We shall give instructions on how to prepare input data for the code, and we shall also explain the various parts of the output produced.

In the following a number of technical LP phrases will be used. They will be explained in later sections, but to help the unprepared user we have compiled a brief LP "vocabulary" in Appendix D.

3.1 Program files

LINPROG communicates with the user via a number of *files*. As a FORTRAN-based system, the code associates an internal unit number with each of these files. The connection between units and files is given in Table 2. The file names are

Table 2. FORTRAN unit numbers for LINPROG files

Unit No.	File	Usage	Format	Name
2	Matrix File	Input	ASCII	matrix.dat
6	Result File	Output	ASCII	result.lis
12	Temporary for Presolve	Scratch	Binary	
15	Restart File	Input	Binary	restart.dat
16	Dump File	Output	Binary	dump.dat
19	Control File	Input	ASCII	control.dat
20	Communication File	Output	Binary	binout.dat

given for typical LINPROG implementations (e.g. PC and Unix), but generally they depend on the specific computer installation. All these files will be described subsequently. They are supposed to reside on the disk storage of the computer, although the Result File was originally designed as a printer file.

The files 2, 6, and 19 are mandatory, while 12, 15, 16, and 20 are optional.

In this context we may also mention the executable file for LINPROG itself. Typically it has the name "linprog.exe" or just "linprog".

For a standard run of LINPROG you need two input files. The first one controls the LP calculation, while the other one contains the constraint matrix. They are described below.

3.2 Preparation of input: Control File

The first input file is called the *Control File*. It contains a verbal instruction set for solving the LP problem(s) in question, with one instruction (command) per

line. Each instruction contains a *keyword*, or a keyword followed by a value. A Control File may be as simple as this:

```
EXEC
PEND
```

A more elaborate example is the following:

```
SOLUTION
MAXITS 10000
EXEC
BINOUT
REDUCE
DUMP
EXEC
PEND
```

The position of a keyword within a line is immaterial, only the line is limited to 72 characters. To be recognized, keywords must be unabridged. The interpretation is case-insensitive. The keywords EXEC and PEND are special in the sense that they control the job stream: EXEC initiates another LP, and PEND tells that there are no more problems to solve. The other keywords are called *numerical* if they are followed by a value, otherwise they are *action* keywords. Commonly, the numerical keywords and the action keywords are called *descriptive* keywords. Between the top of the Control File and the first EXEC, or between two EXECs, the descriptive keywords may come in any order, and this order has no bearing on the order of performing the corresponding LINPROG tasks. Together they specify modes for solving a single problem. All descriptive keywords are optional.

We give below the complete list of action keywords available in the current release of LINPROG:

ECHO	Prints an echo of the Matrix File (Section 3.3)
ELTAB	Prints column and row counts of nonzero LP matrix elements
PICTURE	Prints a picture of the distribution of nonzeros in the LP matrix
REDUCE	Invokes a presolve module, which tries to reduce the LP before the simplex procedure
DUMP	Dumps the basic index set on a file
RESTART	Restarts LINPROG by reading the basic index set from a file
MAX	Tells that this LP is a maximization problem (by default it would be minimization)
SOLUTION	The solution will be printed, both its row and its column part (if omitted, only the optimal value will be printed)
BINOUT	Stores the solution as unformatted (binary) records on an output communication file which can later be read by another program

We have designed the default settings in LINPROG in such a way, that you probably will need the numerical keywords only on rare occasions. They fall in two groups. In the first group the associated values are integers, in the other they are floating-point quantities. First we list the 6 keywords with integral values:

MAXITS	Gives an upper limit of simplex iterations. By default, or by specifying 0, no limit is assumed.
MAXCPM	Gives an upper limit of the CPU time (minutes) allocated to a *single* problem. By default, or by specifying 0, no limit is assumed (the time will be checked after each re-inversion, cf. Section 4.4.)

LOGFRQ Produces a log of the simplex iteration at every LOGFRQ iteration step. By default, or by specifying 0, no iteration printout will be given. Negative values of LOGFRQ are also allowed. In that case |LOGFRQ| gives the frequency of the iteration printout, and in addition a map of the variables in the initial basis is printed (the numbering is the same as for the ROWS and COLUMNS section output explained in Section 3.4). Moreover LOGFRQ < 0 causes a message to be printed at each re-inversion.

MITRE Number of simplex iterations between two consecutive re-inversions (Section 4.4). By default MITRE = 100. Maximum value is MITRE = 1000.

MSCALE Scaling option for constraint matrix.
MSCALE = 0: No scaling
MSCALE = 1: Row scaling
MSCALE = 2: Column scaling
MSCALE = 3: Row and column scaling
By default, MSCALE = 1. The scaling methods are discussed in Section 4.6.5.

NOPROG Gives an upper limit of simplex iterations with no progress in the optimization process. See also the remarks in Section 4.6.4 about cycling. By default, NOPROG = 5000.

The integer specifying the value may be placed anywhere after the keyword up to position 72. For example:

```
MAXCPM    120
          MITRE50
    LOGFRQ -10
```

Here we have set an upper bound of 120 minutes CPU time for the problem, we have changed the re-inversion frequency from 100 to 50 iterations, and we ask for a printout of simplex iterations at every 10 steps, including a map of the initial basis.

The numerical keywords with floating-point values are all used to redefine program tolerances. These are explained in Section 4.6.5, and the default settings are given in Table 3 there. The 9 keywords BIG, EPSCHC, EPSCHR, EPSFEA, EPSINA, EPSLU, EPSPIV, EPSRIN, and ZERPIV are the same as the FORTRAN names in Table 3. New values may be entered in "scientific" notation using the FORTRAN input convention. As an example,

```
    EPSFEA   1.0E-8
```

raises the feasibility tolerance from its default value 10^{-10} to 10^{-8}.

An equal sign is allowed between a numerical keyword and its value, but is not required. For example:

```
EPSCHC = 1.0E-9
MSCALE= 3
```

Finally we repeat the two mandatory keywords used for job control:

EXEC Tells LINPROG to proceed with one LP problem

PEND Tells LINPROG to stop the job stream

3.3 Preparation of input: Matrix File

The second input file is called the *Matrix File*. Its organisation must comply with the Standard MPS format. This format, which is used by most commercial LP systems, was originally developed by IBM, and in the following specifications we shall adhere to the prescriptions given for the MPSX system of IBM (cf. the MPSX manual [5]). As an illustration we show a printout of the Matrix File corresponding to our sample problem:

```
NAME          DIETNAZA
ROWS
 N  COST
 G  PROTEIN
 G  ENERGY
 G  CALCIUM
 G  IRON
 G  VITAMINA
COLUMNS
    POULTRY   COST          0.40   PROTEIN        0.20
    POULTRY   CALCIUM       0.08   IRON          0.014
    POULTRY   VITAMINA      0.80
    SPINACH   COST          0.15   PROTEIN        0.03
    SPINACH   ENERGY        0.03   CALCIUM        0.83
    SPINACH   IRON          0.02   VITAMINA         73
    POTATOES  COST          0.10   PROTEIN        0.02
    POTATOES  ENERGY        0.18   CALCIUM        0.07
    POTATOES  IRON         0.006
RHS
    DEMANDS   PROTEIN        65.0   ENERGY         90.0
    DEMANDS   CALCIUM       200.0   IRON           10.0
    DEMANDS   VITAMINA     5000.0
ENDATA
```

The first record is a NAME line which gives the data set a name. It will appear as an identification of the LINPROG output. The name chosen here is DIETNAZA. It must occupy position 15 – 22. The last record in the file is an ENDATA line which signals the end of the data set. In between there are a number of "sections", each initiated by a headline. There are five possible sections, corresponding to the headlines ROWS, COLUMNS, RHS, RANGES, and BOUNDS, in that order. The first three are mandatory, the last two are optional (neither of them is present in this example). Lines other than headlines contain the problem data. They use six predefined position fields:

Field	Position range	Contents
1	2 – 3	Indicator field (N, E, L, G, UP, LO, etc.)
2	5 – 12	Name field
3	15 – 22	Name field
4	25 – 36	Value field
5	40 – 47	Name field
6	50 – 61	Value field

Names can contain any ASCII character and must not exceed 8 characters in length. Leading blanks of names are significant. Values are numeric and must not exceed 12 characters in length; they may be integral or may contain a decimal point. Even an exponential field is allowed; in fact the numeric input is compatible with formatted FORTRAN inputting.

The ROWS Section: This section is mandatory and defines the rows in the problem. These are the objective function and the problem constraints. In the ROWS Section, a line is given for each row. Such a line contains:

- The restriction type of the row (free in position 2 – 3):
  N = Non-constrained row, usually objective function

E = Equality constraint
L = Less-than-or-equal-to constraint
G = Greater-than-or-equal-to constraint

- The name of the row (in Field 2, position 5 – 12)

The lines in this section may appear in any order. In particular, the objective row need not be the first one. LINPROG requires precisely one object function. The convention is adopted that the *first* row of type N is selected as the object function.

The COLUMNS Section: This section is mandatory. It defines a name for each of the structural variables and lists the *nonzero* entries in the corresponding column position of the objective function and the constraint matrix. All the elements for a column must be grouped together, but they need not appear in the same order as in the ROWS Section. For each line in the COLUMNS Section:

Field 2	contains the name of the variable.
Field 3	contains the name of a row in which the variable has a nonzero coefficient.
Field 4	contains the value of this coefficient.

Fields 5 and 6 are optional. If used, they contain another row name and corresponding coefficient.

The RHS Section: This section is mandatory and deals with the right-hand side of the problem. It gives the right-hand side a name (DEMANDS in our example) and specifies all the nonzero right-hand side values. Except for Field 2, which contains the name of the right-hand side, the fields in the lines of the RHS Section contain the same type of information as the corresponding fields of the COLUMNS Section. LINPROG allows the user formally to specify multiple right-hand sides with different names, but the program can deal with only a single right-hand side and will in that case process the *first* one.

The RANGES Section: This section is optional. It contains the ranges for right-hand side values. To give an example, suppose that we modify the diet problem (2) in Section 2 by adding the constraint that the daily consumption of carbohydrate ("energy") should not exceed 120 g. With the constraint already present this means that the carbohydrate consumption must lie in the interval $[90, 120]$. Instead of adding a new constraint, which would involve more computations, we give only the *range* for the right-hand side. This is done in the RANGES Section as follows:

```
NAME          ...
ROWS
...
COLUMNS
...
RHS
...
RANGES
    RANGE1    ENERGY            30.0
ENDATA
```

In this example, there is one range set name, RANGE1, which appears in Field 2. Field 3 contains the name of the row to which the range applies and Field 4 the value of the range. Note that it is possible to define several ranges for a given problem (LINPROG also allows formally more than one set of ranges with different names, but only the *first* set will be processed by the code).

In general, the RANGES Section is used to define constraints of the form

$$\ell \leq \sum a_j x_j \leq u, \tag{3}$$

by specifying one of the bounds ℓ or u in the RHS Section (as b say), together with a *range value* r in the RANGES Section. This range value may be negative, and LINPROG uses the following (historical) rules for calculating ℓ and u, given b and r:

Type	Sign of r	Lower limit, ℓ	Upper limit, u
E	+	b	$b + \|r\|$
E	−	$b - \|r\|$	b
G	+ or −	b	$b + \|r\|$
L	+ or −	$b - \|r\|$	b

The BOUNDS Section: This section is optional. It defines bounds for the values of the variables. If no bounds are given for a variable, it is assumed to have a lower bound of zero and no upper bound. To give an example, suppose that we modify the diet problem (2) in Section 2 by adding the constraint that the daily poultry supply is limited to 200 g per inhabitant. We may then add the constraint POULTRY ≤ 200 via a BOUNDS Section input, which is cheaper than expressing the bound as an additional constraint row. We then get the following modification of the Matrix File:

```
NAME          ...
ROWS
...
COLUMNS
...
RHS
...
BOUNDS
 UP LIMIT     POULTRY           200
ENDATA
```

In general, each data line in the BOUNDS Section defines a bound for one variable; a number of bound types other that UP may be imposed. For each line in the BOUNDS Section:

Field 1 — specifies the type of bound:
UP for an upper bound, $0 \leq x \leq u$.
LO for a lower bound, $x \geq \ell$.
FX for a fixed-variable bound, $x = a$.
FR for a free variable, $-\infty < x < +\infty$.
PL for a nonnegative variable, $0 \leq x < +\infty$ (default).
MI for a nonpositive variable, $-\infty < x \leq 0$.

Field 2 — defines the name of the bound set.

Field 3 — defines the name of the variable to which the bound applies.

Field 4 — specifies the value of the bound (omitted for FR, PL, and MI).

In our example there is only one bound set called LIMIT containing a single bound. It is possible, however, to define several bounds for a given bound set (LINPROG also allows formally more than one set of bounds with different names, but only the first set will be processed by the code.) By using both LO and UP you may specify two-sided bounds like $\ell \leq x \leq u$.

Remember that if both a RANGES and a BOUNDS Section are present, RANGES must come before BOUNDS.

The Matrix File may contain more than one data set. Each data set is initiated by a NAME line and ended by an ENDATA line; the number of data sets should match the number of EXEC lines in the Control File.

3.4 Interpretation of output

We shall now describe the *Result File* printed by LINPROG. Our first sample output is from the diet problem without ranges and bounds. It looks as follows:

```
#########################################################################
#                        LINPROG VERSION 9501                           #
#                   LINEAR PROGRAMMING CODE WRITTEN BY                  #
#               PETER KIRKEGAARD AND POUL ERIK GROHNHEIT                #
#         RISOE NATIONAL LABORATORY, DK-4000 ROSKILDE, DENMARK          #
#                          COPYRIGHT (C) 1995                           #
#########################################################################


 DATE AND TIME OF PROGRAM RUN: 95/01/26 13:09   (DAY NUMBER 95/026)

 LINPROG 9501 COMPILATION OF JOB FROM COMMAND FILE

         SOLUTION

 NAME OF DATA SET: DIETNAZA

  PROBLEM STATISTICS

      6 LP ROWS       9 VARIABLES      22 LP ELEMENTS  DENSITY =  40.74

  THESE STATISTICS CONTAIN ONE SLACK VARIABLE FOR EACH ROW

                        TOTAL  NORMAL  .FREE.   FIXED BOUNDED
  ROWS     (LOG.VAR.)       6       5       1       0       0
  COLUMNS  (STR.VAR.)       3       3       0       0       0

 PROGRAM TOLERANCES:
 BIG    = 1.0E+30     EPSCHC= 1.0E-10     EPSCHR= 1.0E-10
 EPSFEA= 1.0E-10      EPSINA= 1.0E-08     EPSLU = 1.0E-13
 EPSPIV= 1.0E-09      EPSRIN= 1.0E-02     ZERPIV= 1.0E+00

 OTHER PARAMETERS:
 LOGFRQ=     0        MAXCPM=     0       MAXITS=     0
 MITRE =   100        MSCALE=     1       NOPROG= 5000

 END OF PHASE 1  (ESTABLISHMENT OF FEASIBILITY)
 ITERATION COUNT:       1        TIME (SEC):       0.

 END OF PHASE 2

 SOLUTION   (OPTIMAL)

          ...NAME...     ...ACTIVITY...       DEFINED AS

          FUNCTIONAL        174.70817        COST
          RESTRAINTS                         DEMANDS


 LINPROG 9501 EXECUTION

 SECTION 1 - ROWS


  NUMBER  ...ROW..  AT  ...ACTIVITY...  SLACK ACTIVITY  ..LOWER LIMIT.  ..UPPER LIMIT.  .DUAL ACTIVITY

       1  COST      BS      174.70817       174.70817-          NONE           NONE         1.00000
       2  PROTEIN   LL       65.00000            .           65.00000          NONE         1.67315-
       3  ENERGY    LL       90.00000            .           90.00000          NONE         0.21401-
       4  CALCIUM   BS      205.49125         5.49125-      200.00000          NONE          .
       5  IRON      LL       10.00000            .           10.00000          NONE         4.66926-
       6  VITAMINA  BS    13621.59533      8621.59533-     5000.00000          NONE          .


 LINPROG 9501 EXECUTION

 SECTION 2 - COLUMNS
```

```
 NUMBER  .COLUMNS  AT  ...ACTIVITY...  ..INPUT COST..  ..LOWER LIMIT.  ..UPPER LIMIT.  .REDUCED COST.

      7  POULTRY   BS      250.48638        0.40000          .              NONE          .
      8  SPINACH   BS      183.85214        0.15000          .              NONE          .
      9  POTATOES  BS      469.35798        0.10000          .              NONE          .

MAX VIOLATION OF RESTRAINTS WAS  1.776E-15
IT OCCURRED IN THE RESTRAINT     PROTEIN    G   6.500E+01

ITERATION COUNTS IN PHASES  0, 1, 2:         0       1       2

TIME FOR THIS JOB:     0.03 SECONDS.

DATE AND TIME OF PROGRAM RUN: 95/01/26 13:09   (DAY NUMBER 95/026)

LINPROG 9501 COMPILATION OF JOB FROM COMMAND FILE


                          OUTPUT SUMMARY
  CASE    ROWS COLS  ELEM  PH.0 PH.1 ITNS  VIOL         OPTIMUM        CPU-SEC
DIETNAZA     6    3    16    0    1     3 1.8E-15  1.7470817120623E+02    0.03

                                                  GRAND TOTAL TIME     0.07
```

Our second sample output is from the modified diet problem with RANGES and BOUNDS. To save space we print only the solution part for this problem:

```
.........
SOLUTION   (OPTIMAL)

         ...NAME...     ...ACTIVITY...       DEFINED AS

         FUNCTIONAL        205.00000        COST
         RESTRAINTS                         DEMANDS
         BOUNDS....                         LIMIT
         RANGES....                         RANGE1


LINPROG 9501 EXECUTION

SECTION 1 - ROWS

 NUMBER   ...ROW..  AT  ...ACTIVITY...  SLACK ACTIVITY  ..LOWER LIMIT.  ..UPPER LIMIT.  .DUAL ACTIVITY

      1   COST      BS      205.00000      205.00000-          NONE           NONE         1.00000
      2   PROTEIN   LL       65.00000           .            65.00000        NONE         5.00000-
A     3   ENERGY    LL       90.00000           .            90.00000      120.00000          .
      4   CALCIUM   BS      511.31250      311.31250-       200.00000        NONE          .
      5   IRON      BS       16.48750        6.48750-        10.00000        NONE          .
      6   VITAMINA  BS    41222.50000    36222.50000-      5000.00000        NONE          .


LINPROG 9501 EXECUTION

SECTION 2 - COLUMNS

 NUMBER  .COLUMNS  AT  ...ACTIVITY...  ..INPUT COST..  ..LOWER LIMIT.  ..UPPER LIMIT.  .REDUCED COST.

      7  POULTRY   UL      200.00000        0.40000          .          200.00000        0.60000-
      8  SPINACH   BS      562.50000        0.15000          .              NONE          .
      9  POTATOES  BS      406.25000        0.10000          .              NONE          .
.........
```

The output from LINPROG begins with a heading followed by a printout of the lines of the Control File for the present LP problem, and the data set name defined in the Matrix File. Next comes some problem statistics in the same format as for example MPSX [5] uses. In the present case we have 5 normal rows and 1 free row. The phrases "normal" and "free" mean inequality-constrained and unconstrained,

respectively. Our example has the objective row as the only unconstrained row. After a list of numerical parameter settings, information about the various phases of the simplex procedure is printed. This part of the output is usually unimportant in standard runs of LINPROG. In any case, its interpretation becomes clear if you consult the subsequent sections of this report. Notice, however, the message "ESTABLISHMENT OF FEASIBILITY" (after END OF PHASE 1) telling that our LP is indeed feasible; if this were not the case a message "THIS PROBLEM HAS NO FEASIBLE SOLUTION" would be issued. After this comes the solution printout summary. First there is a heading part containing

- The name SOLUTION with a qualifying text (OPTIMAL or INFEASIBLE)
- The value and name of the objective function
- The name of the right-hand side. If bounds and ranges sets are present, the name of these.

After the heading we find the ROWS Section and COLUMNS Section output (but only if the keyword SOLUTION occurs in the Control File). The two sections have similar structure and are formatted in the same way as the MPSX code [5]. Each section is printed as a table of 8 columns, where one row of the table corresponds to a row (or column) of the constraint matrix. Each row and column in this matrix is formally associated with a variable and given a name. Below we give a short description of the items contained in the 8 columns:

1. NUMBER — The sequential number given by LINPROG to the row or column. If the problem contains m rows and n columns, the rows are numbered from 1 through m and the columns from $m+1$ through $m+n$. In our two examples, the numbers of row VITAMINA and SPINACH are 6 and 8, respectively.
2. NAME — This is the 8-character name given to the row or column.
3. AT — This is a two-character code denoting the status that the row or column variable has in the solution. The codes and their meanings are:

   BS — basic and feasible

   EQ — nonbasic and fixed

   UL — nonbasic at upper limit

   LL — nonbasic at lower limit

   ** — infeasible

   There are two different nonbasic status indicators UL and LL, because the LP variables may be upper- and lower-bounded (see also Section 4.6.1).
4. ACTIVITY — This is the value that the row or column variable takes in the solution:
   - For columns, the activity is simply the value given to the column in the solution. For example, the value of POULTRY is 250.48638 (in gram units).
   - For rows, the activity is defined as the sum of the products of constraint coefficients and the column activities. For example, the row ENERGY, which defines the constraint (cf. (2)) $0.03 \times \text{SPINACH} + 0.18 \times \text{POTATOES} \geq 90$, has the activity $0.03 \times 183.85214 + 0.18 \times 469.35798 = 90$.

5. SLACK ACTIVITY (ROWS Section) and INPUT COST (COLUMNS Section) — For a given row, the "slack activity" is the difference between the right-hand side element for the row and the activity of the row. For a column, the "input cost" is the corresponding element in the objective function.

6. LOWER LIMIT — The lower bound on the row or column activity.

7. UPPER LIMIT — The upper bound on the row or column activity.

8. DUAL ACTIVITY (ROWS Section) — The value given under "dual activity" is meaningful only for rows at one of its bounds. If such a bound could be removed, we would expect a decrease of the total cost, and the dual activity gives the cost reduction per unit relaxation of the limit. In our example the diet must contain at least 65 g protein. In fact, the optimal solution contains precisely 65 g. If the required minimum of 65 g could be lowered by 1 g, the dual activity indicates a saving of 1.67 cents. But notice that this figure is actually a rate of change, valid close to the solution. Anyway, the dual activity can help to suggest alterations of the specified model constraints.
   REDUCED COST (COLUMNS Section) — For a given column, the "reduced cost" is a quantity with a similar meaning as the dual activity for a row. Consider for example the output of the second problem, where POULTRY has an activity of 200 (gram) in the optimal solution. Its reduced cost is -0.6, which indicates that if one gram of POULTRY were withdrawn from the diet, a cost reduction of 0.6 cents would be gained (in general the reduced cost is the *decrease* in objective function per unit *increase* of the variable; in our case POULTRY is at its upper bound and can only be decreased, which explains the negative sign of the reduced cost).

Both dual activity and reduced cost can be thought of as "unit costs", the former valid for a row, the latter for a column. Dual activities are sometimes called "shadow prices" or "simplex multipliers". Notice that there is some ambiguity in the sign definition of dual activities among commercial LP systems. We follow here MPSX [5], but MINOS [6] has the opposite sign definition.

For historical reasons we also follow the MPSX convention of presenting negative quantities by using postfix sign notation.

Sometimes an "A" is printed in front of the line corresponding to a row or column variable (we see an instance of this for the variable ENERGY in our second problem). The meaning of this "A" is that there is a possibility of *alternative* optimal solutions, in which the indicated variable (and other variables) might take different values.

An infeasible solution will not prevent LINPROG from printing the ROWS and COLUMNS output sections. As mentioned previously, the status of each infeasible row or column is then printed as **. In addition LINPROG prints a table of all the infeasible rows. But before drawing conclusions about the origin of the infeasibility, the user should observe that the printout is only a snapshot of the variables at the time LINPROG decided the infeasibility.

After the ROWS and COLUMNS Section output LINPROG concludes the problem by printing the detected maximum violation of constraints and the CPU-time consumption.

As pointed out earlier there may be more than one problem in a LINPROG job stream (these test examples have only one). After the output for all the individual problems, LINPROG concludes by printing an OUTPUT SUMMARY with a brief record of key data for each problem. A summary line contains 10 items with the following contents:

CASE: Name of the data set.

ROWS:	Number of rows in the LP, including the objective row.
COLS:	Number of structural columns in the LP.
ELEM:	Number of nonzero elements in the constraint matrix. The objective row is included, but slack elements are excluded.
PH.0:	Number of eliminated zero slacks in Phase 0 (Section 4.6.2).
PH.1:	Number of iterations in Phase 1 (Section 4.6.3).
ITNS:	Total iteration count. This is the sum of PH.0, PH.1 and (if feasibility was achieved) the number of iterations in Phase 2.
VIOL:	Maximum detected constraint violation. For an infeasible problem asterisks are printed.
OPTIMUM:	The optimal value. For an infeasible problem asterisks are printed.
CPU-SEC:	The CPU time for the problem in seconds.

If presolve (Section 4.7) was activated by the REDUCE keyword, the output summary will show a P after ELEM, and if the job is a RESTART job (Section 3.5), an R is printed after VIOL.

If an error condition occurs during the LP computation, LINPROG will print a message. Numerous error conditions are possible, but in any case the error message should be self-explanatory.

3.5 Dump/restart facility

LINPROG has a dump/restart facility, which involves the optional files 15 and 16, cf. Table 2. If the keyword DUMP is specified in the Control File, LINPROG will conclude the execution by a dump on unit 16. This dump does not contain the solution itself, but the so-called *basic index set*, from which the solution can be retrieved by a comparatively cheap *inversion* process. Not only is a dump made at the end of a normal successful execution, but dumps are also made at intermediate points of the computations (at times of re-inversion, see Section 4.4), such that each dump overwrites the previous one. In a subsequent run you can specify RESTART in the Control File and use the previous dump file as a restart file after a proper renaming. The dump/restart facility is useful in the following situations:

- You may prevent a long LINPROG run from being wasted due to unforeseen failure such as the discontinuation of a job, if some resource limit is exceeded.
- You can divide very time-consuming LP runs into manageable portions.
- You may use DUMP/RESTART for a series of similar problems, where you want to change some coefficients in the LP matrix or right-hand side, provided you don't change the number of restraints or variables.

Restart files are checked for compatibility by LINPROG; for possible conflicts with presolve, see Section 4.7.

In dump/restart files the information is stored in binary (unformatted) form.

It is natural to use DUMP, whenever the keywords MAXCPM or MAXITS (Section 3.2) are in use.

For safety reasons there will be an OPEN and a CLOSE of the DUMP file for each DUMP. This precaution, which was proposed by VTT in Finland, ensures that the DUMP file is left intact should the program be interrupted by the user or otherwise be disconnected.

3.6 Binary output and report writers

In large-scale applications of linear programming the input data to the LP solver is normally produced by a special computer program, tailored to the application. Such a program is called a *matrix generator*. Likewise, there will often be a *report-writer* program which reads the LP output and presents the results in the form of edited and formatted tables suited to the particular problem. One way to design a report writer for LINPROG would be to let it read the ASCII Result File, which of course should be directed to the disk for such a purpose. The advantage of this method is that it is fairly general: An ASCII file can be read by a report writer coded in any programming language that suits your purpose. It can be inspected directly and be transferred between different computer systems.

Alternatively, you may let LINPROG produce a binary (i.e. unformatted) *communication file* containing the solution. To do this you include the keyword BINOUT in the Control File (cf. Section 3.2). The binary output file will be connected to unit 20 in LINPROG (cf. Table 2). The merits of the communication file is that it holds the solution in full machine precision; there is no rounding error due to decimal formatting. It needs less disk space than the ASCII-formatted Result File, and both writing and reading of unformatted records are faster than for formatted records.

The LINPROG communication file consists of multiple solution sets, if more than one LP was solved using the BINOUT keyword. A solution set contains an identification section, a row section, and a column section. The precise contents of each are stated in the following record tables.

The identification section has 3 records:

Record # 1		
Item #	Type	Contents
1	CHARACTER*6	Date of run in the form YYMMDD
2	CHARACTER*4	LINPROG version in the form YYMM
3	CHARACTER*8	'MINIMIZE' or 'MAXIMIZE'

Record # 2		
Item #	Type	Contents
1	CHARACTER*8	Name of the data set
2	CHARACTER*8	Name of the objective row
3	CHARACTER*8	Name of the right-hand side
4	CHARACTER*8	Name of RANGES
5	CHARACTER*8	Name of BOUNDS

Record # 3		
Item #	Type	Contents
1	CHARACTER*1	'O' for optimal, 'I' for infeasible.
2	INTEGER	Number of rows, L.
3	INTEGER	Number of columns, NS.
4	INTEGER	Iteration counter
5	DOUBLE PRECISION	Function value

The row section has L records, one for each row:

Record # I (I = 1 , . . . , L)		
Item #	Type	Contents
1	CHARACTER*1	'A' if alternative solution, otherwise blank
2	CHARACTER*8	Name of the row
3	CHARACTER*2	Status of the row
4	DOUBLE PRECISION	Activity of the row
5	DOUBLE PRECISION	Slack activity of the row
6	DOUBLE PRECISION	Dual activity

The column section has NS records, one for each column:

Record # J (J = 1 , . . . , NS)		
Item #	Type	Contents
1	CHARACTER*1	'A' if alternative solution, otherwise blank
2	CHARACTER*8	Name of the column
3	CHARACTER*2	Status of the column
4	DOUBLE PRECISION	Activity of the column
5	DOUBLE PRECISION	Input cost
6	DOUBLE PRECISION	Reduced cost

The items in the row and column sections correspond to the items explained for the printed output in Section 3.4. However, we do not give the number of the variable, nor its bounds, in the binary output. Under the heading "Type" we specify the items using phrases from the FORTRAN language. CHARACTER*n means a text of length n bytes. The numerical types are INTEGER and DOUBLE PRECISION; for many computers these will use 4 and 8 bytes of storage, respectively, by default.

Appendix A gives a printout of a paragon form of a report-writer program. This program, REPORT, reads the communication file and prints first a summary for the LINPROG run and then the two sections of the solution. It also stores some of the variables in arrays, anticipating some subsequent post-processing, depending on the actual application.

As a test of the REPORT program, we repeated our second LINPROG test run (the diet problem with RANGES and BOUNDS), this time with a BINOUT keyword in the Control File. Taking the binary LINPROG output as input, REPORT produced the following output:

```
PROBLEM DATE                  950130
LINPROG VERSION               9501
TARGET                        MINIMIZE

NAME OF DATA SET              DIETMODI
NAME OF OBJECTIVE ROW         COST
NAME OF RIGHT-HAND SIDE       DEMANDS
NAME OF RANGES                RANGE1
NAME OF BOUNDS                LIMIT

PROBLEM STATUS                OPTIMAL
NUMBER OF ROWS                   6
NUMBER OF COLUMNS                3
NUMBER OF ITERATIONS             2
OBJECTIVE VALUE                2.0500000000000E+02

TABULATION OF THE FILED ROW SECTION
 NAME     NUMBER STATUS   ACTIVITY        SLACK       DUAL ACTIVITY   MARK
COST          1    BS    2.050000E+02 -2.050000E+02   1.000000E+00
PROTEIN       2    LL    6.500000E+01  0.000000E+00  -5.000000E+00
ENERGY        3    LL    9.000000E+01  0.000000E+00   0.000000E+00    A
CALCIUM       4    BS    5.113125E+02 -3.113125E+02   0.000000E+00
IRON          5    BS    1.648750E+01 -6.487500E+00   0.000000E+00
VITAMINA      6    BS    4.122250E+04 -3.622250E+04   0.000000E+00

TABULATION OF THE FILED COLUMN SECTION
 NAME     NUMBER STATUS   ACTIVITY      INPUT COST    REDUCED COST   MARK
POULTRY       7    UL    2.000000E+02  4.000000E-01  -6.000000E-01
SPINACH       8    BS    5.625000E+02  1.500000E-01   0.000000E+00
POTATOES      9    BS    4.062500E+02  1.000000E-01   0.000000E+00
```

4 Mathematical description

This chapter serves as a mathematical documentation of the theory and methods applied to LINPROG. Readers with no interest in such a documentation may wish to skip the entire chapter, or read Section 4.1 only.

4.1 The simplex method: An overview

We shall here give a short review of the simplex method in the form we use it in LINPROG. A more detailed discussion of the various components of the algorithm is postponed to later sections. Let us also mention that there are many good textbooks dealing with simplex. First there is Dantzig's classical book [7], and among the newer books we could refer to Murtagh [8] and Nazareth [4].

4.1.1 Standard form of the LP

We begin with a reformulation of our LP using vector-matrix notation:

$$\text{Minimize} \quad z = \mathbf{c}^{\mathrm{T}}\mathbf{x} \tag{4}$$

$$\text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b} \tag{5}$$

$$\text{and} \quad \mathbf{x} \geq \mathbf{0}. \tag{6}$$

Here and in the following we use bold-face small and capital letters to denote vectors and matrices, respectively, like $\mathbf{c} = \{c_j\}$, $\mathbf{x} = \{x_j\}$ and $\mathbf{A} = \{a_{ij}\}$. A superscript-T stands for matrix or vector transpose, such that e.g. $\mathbf{c}^{\mathrm{T}}\mathbf{x}$ becomes a scalar product $\mathbf{c} \cdot \mathbf{x}$.

It is clear that our original LP formulation (1) in Chapter 2 can always be brought to this standard form. We could first convert all $\geq$-rows in (1) to $\leq$-rows by multiplication by -1. Next the $\leq$-rows could be transformed to equalities by introducing nonnegative *slack variables*; obviously an inequality of the form $\sum a_j x_j \leq b,\ x_j \geq 0$ is equivalent to an equality $s + \sum a_j x_j = b,\ s \geq 0, x_j \geq 0$, where s is a slack variable. The slack variables contribute to $\mathbf{A}$ in (5) with a subset of the columns of the unit matrix $\mathbf{I}$. Rows with dummy restrictions $\mathcal{R}_i$ in (1) are simply deleted from the LP. Assuming these operations to be already done, our LP has now n variables in total, and m restrictions. Thus the *constraint matrix* $\mathbf{A}$ becomes an $m \times n$ matrix. This matrix is supposed to have full rank, $\rho(\mathbf{A}) = m$, that is, the restrictions are independent, so we have $n \geq m$. Moreover, $\mathbf{c}$ and $\mathbf{x}$ are n-vectors with an inner product equal to the *objective function* z in (4), and $\mathbf{b}$ is an m-vector.

LINPROG is able to deal with more general problem formulations than envisaged in (1) or (4) – (6), but we lose little in referring to the standard form to explain our use of simplex. For example, maximizing $\mathbf{c}^{\mathrm{T}}\mathbf{x}$ is equivalent to minimizing $-\mathbf{c}^{\mathrm{T}}\mathbf{x}$. LINPROG can also treat inhomogeneous objective functions $c_0 + \mathbf{c}^{\mathrm{T}}\mathbf{x}$. A less trivial extension is the capability of the program to handle bounded variables and range constraints. We shall describe later (Section 4.6.1) how these facilities are implemented, using a modification of the simplex method. Another point to mention is the ability of LINPROG to cope with a rank-deficient constraint matrix: The program detects redundant or conflicting restrictions, and by dispensing of the superfluous equations it produces a reduced matrix with full row rank.

4.1.2 The basis exchange mechanism

Let now a subset of m independent columns of $\mathbf{A}$ be given. Arranged in any order they form a square nonsingular matrix $\mathbf{B}$, which we call a *basis matrix*. The

corresponding variables x_j are called the *basic* variables; the remaining variables are said to be *nonbasic*. The fundamental idea in the simplex method is to operate with solutions $\mathbf{x} = \{x_j\}$ in which the $n - m$ nonbasic components are 0, while the m basic variables may be nonzero (some basic variables may be zero, too, and then we have a *degenerate* solution). If a basic solution $\mathbf{x} = \{x_j\}$ satisfies (5) and (6), we call it *basic feasible*. If (5) is satisfied but not (6) we have an *infeasible* "solution". The partition of the variables in basic and nonbasic described here is dynamic in the sense that the simplex process in each iteration step exchanges the state of two variables (x_j, x_k) such that x_j "enters the basis" and x_k "leaves the basis", that is, it becomes zero. In each step we move from one basic feasible solution to another with a better (at least not worse) objective function. In practice we reach the optimum in a finite number of steps; see however the comments given in Section 4.6.4 on the possibility of "cycling". By the end of the exchange step, $\mathbf{B}$ is transformed to an *adjacent* basis matrix $\overline{\mathbf{B}}$. In order that we can get the simplex method to work, we must provide an initial basic feasible solution. This is achieved by a special technique to be discussed in Section 4.6.3.

Let us take a closer look of what happens in an exchange step. If we renumber the columns in $\mathbf{A}$ such that the current basic columns precede the nonbasic columns, we may write $\mathbf{A}$ as a partitioned matrix

$$\mathbf{A} = (\mathbf{B} \mid \mathbf{N}), \tag{7}$$

with a corresponding partitioning and ordering of $\mathbf{x}$ and $\mathbf{c}$,

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_\mathrm{B} \\ \mathbf{x}_\mathrm{N} \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} \mathbf{c}_\mathrm{B} \\ \mathbf{c}_\mathrm{N} \end{pmatrix}, \tag{8}$$

while (5) becomes

$$\mathbf{B}\mathbf{x}_\mathrm{B} + \mathbf{N}\mathbf{x}_\mathrm{N} = \mathbf{b}. \tag{9}$$

The components of the two parts of the solution vector $\mathbf{x}$ are

$$(\mathbf{x}_\mathrm{B})_i = x_i \quad (i = 1, \ldots, m), \qquad (\mathbf{x}_\mathrm{N})_j = x_{m+j} \quad (j = 1, \ldots, n - m); \tag{10}$$

Similarly for the cost vector $\mathbf{c}$:

$$(\mathbf{c}_\mathrm{B})_i = c_i \quad (i = 1, \ldots, m), \qquad (\mathbf{c}_\mathrm{N})_j = c_{m+j} \quad (j = 1, \ldots, n - m); \tag{11}$$

The ordering of columns within $\mathbf{B}$ (and within $\mathbf{N}$) is immaterial here (a discussion of how to number the rows and columns in $\mathbf{B}$ will be given later). Following common practice we introduce the *transformed right-hand* vector

$$\boldsymbol{\beta} = \mathbf{B}^{-1}\mathbf{b} \tag{12}$$

and the *pricing vector*

$$\boldsymbol{\pi}^\mathrm{T} = \mathbf{c}_\mathrm{B}^\mathrm{T}\mathbf{B}^{-1}, \tag{13}$$

whose components are called *simplex multipliers*. Then we obtain the following problem formulation in terms of the nonbasic variables:

$$\text{Minimize} \quad z = \mathbf{c}_\mathrm{B}^\mathrm{T}\boldsymbol{\beta} + (\mathbf{c}_\mathrm{N}^\mathrm{T} - \boldsymbol{\pi}^\mathrm{T}\mathbf{N})\mathbf{x}_\mathrm{N} \tag{14}$$

$$\text{subject to} \qquad \mathbf{x}_\mathrm{B} = \boldsymbol{\beta} - \mathbf{B}^{-1}\mathbf{N}\mathbf{x}_\mathrm{N} \tag{15}$$

$$\text{and} \qquad \mathbf{x}_\mathrm{B}, \mathbf{x}_\mathrm{N} \geq \mathbf{0}. \tag{16}$$

The current basic solution is obtained by letting $\mathbf{x}_\mathrm{N} = \mathbf{0}$, thus

$$\mathbf{x}_\mathrm{B} = \boldsymbol{\beta} \geq \mathbf{0}, \tag{17}$$

assuming basic feasibility. But (14) and (15) tell more, since the $(n - m)$-vector $\mathbf{d} = \{d_j\}$ defined by

$$\mathbf{d}^\mathrm{T} = \mathbf{c}_\mathrm{N}^\mathrm{T} - \boldsymbol{\pi}^\mathrm{T}\mathbf{N} \tag{18}$$

governs the increase of the objective function when the nonbasic variables rise from their zero bounds. The d_j are called *reduced costs*. Incidentally, $\mathbf{d}$ is the gradient vector of the objective function in the current space of nonbasic variables.

Assume now that $d_j < 0$ for some j, say $j = q$. We may then introduce the qth nonbasic variable $(\mathbf{x}_N)_q = x_{m+q}$ into the basis. This means that we increase this variable by a certain amount θ, while all the other components of $\mathbf{x}_N$ are kept at zero. This will cause z to decrease with the linear slope $|d_q|$. But how large a step θ can we take without destroying the feasibility? This question is answered by considering (15), which for the present case specializes to

$$\mathbf{x}_B = \boldsymbol{\beta} - x_{m+q}\boldsymbol{\alpha}_q, \tag{19}$$

where we use $\boldsymbol{\alpha}_j$ to denote the jth transformed nonbasic column vector of $\mathbf{A}$,

$$\boldsymbol{\alpha}_j = \mathbf{B}^{-1}\mathbf{a}_{m+j}, \tag{20}$$

still obeying the column numbering of $\mathbf{A}$ laid down by (7). If $\boldsymbol{\beta} = \{\beta_i\}$ and $\boldsymbol{\alpha}_q = \{\alpha_{iq}\}$, we find θ by the classical *ratio test*

$$\theta = \min\{\beta_i/\alpha_{iq} : i = 1, \ldots, m \text{ and } \alpha_{iq} > 0\} \tag{21}$$

If no element $\alpha_{iq} > 0$ exists, we have detected an *unbounded solution*, $\theta = \infty$. Otherwise let the minimum in (21) occur for $i = p$. When the corresponding step $x_{m+q} = \theta$ is taken, the pth basic variable reaches zero first and is selected to leave the basis. The number p is called the *pivot index*, while α_{pq} is called the *pivot element*. In the degenerate case the step θ will be 0, and we get no improvement in the objective function.

It is instructive to look at the simplex method from a geometrical viewpoint. The set of all feasible vectors $\mathbf{x}$, i.e. those satisfying the constraints (5) and (6), forms a polytope ("simplex") in n-dimensional space R^n. This polytope may be null, but otherwise one of its vertices is a minimizing point for the objective function z in (4). In the simplex method we carry out a systematic search of the vertices. At each point the $n - m$ nonbasic variables take the value zero, while the m basic variables are determined by (17) and (12). The search proceeds from vertex to adjacent vertex along an edge where a single nonbasic variable for the first vertex increases from zero. The edges are chosen so that the objective function decreases. Since the polytope has only a finite number of vertices, the simplex algorithm is finite unless cycling (Section 4.6.4) occurs.

4.1.3 The simplex algorithm

We are now ready to present the main steps of the simplex algorithm. The steps are as follows:

- INITIALIZATION Step — Begin with a basic feasible solution (17) (Section 4.6.3).
- BTRAN Step — Compute the simplex multipliers by (13).
- PRICE Step — Compute the reduced costs $\{d_j\}$ for the nonbasic variables by (18).
- CHUZC Step — Choose the entering nonbasic variable ($j = q$) as one with a negative reduced cost d_j. (A common rule is to choose one with maximum $|d_j|$; see also Section 4.6.4). If there is no negative reduced cost, the current solution is optimal.
- FTRAN Step — Transform the entering column by (20).

- CHUZR Step — Use the ratio test (21) to find a step θ. This test results either in a variable to leave the basis (pivot index $i = p$), or in an unbounded solution with $\theta = \infty$.
- PIVOT Step — In the basis matrix $\mathbf{B}$, replace the column associated with the leaving variable with that corresponding to the entering one, to obtain an adjacent basis matrix $\overline{\mathbf{B}}$. Update the β-vector defined in (12). Return to the BTRAN step.

The above acronyms are well-established abbreviations for key operations in linear programming: BTRAN means "backward transformation" (postmultiplication by $\mathbf{B}^{-1}$), CHUZC means "choose a column", FTRAN means "forward transformation" (premultiplication by $\mathbf{B}^{-1}$), and CHUZR means "choose a row".

If we were to solve only small problems, it would be adequate to maintain the so-called *current tableau* $\mathbf{B}^{-1}\mathbf{A}$. That method could be called "direct simplex". In *revised simplex*, which is used here as well in the majority of simplex codes, we maintain only the basis-inverse $\mathbf{B}^{-1}$ itself. The advantages of this procedure are connected to the use of a *product representation* of $\mathbf{B}^{-1}$, which we discuss in Section 4.3. The product form is expanded each time a simplex step is executed, and eventually we need to shorten the representation; this is done by the so-called *re-inversion* process described in Section 4.4.

In our list of simplex operations the PIVOT Step deals with the updating mechanism when passing from one basis matrix to an adjacent one. This step, which involves a good deal of computation, exploits the Forrest-Tomlin method described at length in Section 4.5.

4.2 Useful matrix relations

At this place we recapitulate some well-known results from linear algebra, since they will be needed in the following. We shall concentrate on square matrices; all the considered matrices are assumed to be of the order m and nonsingular, hence invertible.

4.2.1 The Sherman-Morrison identity

We begin with a useful formula for inverting "rank-one" updated matrices:

Proposition 1 (Sherman-Morrison) *Given vectors* $\mathbf{u}$ *and* $\mathbf{v}$ *in* R^m, *and the nonsingular matrix* $\mathbf{A} \in \mathsf{R}^{m\times m}$. *If* $\mathbf{v}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{u} \neq -1$, *then*

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^{\mathrm{T}})^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^{\mathrm{T}}\mathbf{A}^{-1}}{1 + \mathbf{v}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{u}} \tag{22}$$

This result, given for example in Golub and van Loan [9], and in Murtagh [8], is easily proved by a direct check.

4.2.2 Elementary matrices

An *elementary column matrix* $\mathbf{E}$ is one which differs from the unit matrix $\mathbf{I}$ in just one column. If we denote the pth unit vector by $\mathbf{e}_p$, such a matrix can be written

$$\mathbf{E} = (\mathbf{e}_1, \ldots, \mathbf{e}_{j-1}, \alpha, \mathbf{e}_{j+1}, \ldots, \mathbf{e}_m), \tag{23}$$

or

$$\mathbf{E} = \begin{pmatrix} & & j & \\ 1 & & \alpha_1 & \\ & \ddots & \vdots & \\ & & \alpha_j & \\ & & \vdots & \ddots \\ & & \alpha_m & & 1 \end{pmatrix} \tag{24}$$

If we want to emphasize that we have inserted a column vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)^{\mathrm{T}}$ at the jth column place, we may write

$$\mathbf{E} = \mathbf{E}_j(\boldsymbol{\alpha}). \tag{25}$$

Elementary column matrices are sometimes called Frobenius matrices. We may write $\mathbf{E}$ in the form of a rank-one update of $\mathbf{I}$:

$$\mathbf{E} = \mathbf{I} + (\boldsymbol{\alpha} - \mathbf{e}_j)\mathbf{e}_j^{\mathrm{T}} \tag{26}$$

Using the Sherman-Morrison identity (22), or by direct means, we find that the inverse is another elementary column matrix,

$$\mathbf{E}^{-1} = \mathbf{I} - \frac{1}{\alpha_j}(\boldsymbol{\alpha} - \mathbf{e}_j)\mathbf{e}_j^{\mathrm{T}} = \begin{pmatrix} & & j & \\ 1 & & -\alpha_1/\alpha_j & \\ & \ddots & \vdots & \\ & & 1/\alpha_j & \\ & & \vdots & \ddots \\ & & -\alpha_m/\alpha_j & & 1 \end{pmatrix}. \tag{27}$$

Hence

$$(\mathbf{E}_j(\boldsymbol{\alpha}))^{-1} = \mathbf{E}_j(\boldsymbol{\eta}), \tag{28}$$

where

$$\boldsymbol{\eta} \equiv \mathrm{g}_j(\boldsymbol{\alpha}) = \{\eta_{ij}\}, \qquad \eta_{ij} = -\alpha_i/\alpha_j, i \neq j, \quad \eta_{jj} = 1/\alpha_j. \tag{29}$$

Also in this context we call the jj-entry α_j of $\mathbf{E}$ in (24) the *pivot element*, and column j is called the *pivot column*.

In complete analogy with elementary column matrices we speak of *elementary row matrices*. Such a matrix differs from $\mathbf{I}$ in just one row. In this work we shall use only elementary row matrices with a unit pivot element:

$$\mathbf{F} = \mathbf{F}_i(\boldsymbol{\gamma}) = \begin{matrix} \\ \\ i \\ \\ \\ \end{matrix} \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ \gamma_1 & \ldots & 1 & \ldots & \gamma_m \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}, \tag{30}$$

where $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_{i-1}, 1, \gamma_{i+1}, \ldots, \gamma_m)$. The inverse of (30) is another elementary row matrix

$$\mathbf{F}^{-1} = \begin{matrix} \\ \\ i \\ \\ \\ \end{matrix} \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ -\gamma_1 & \ldots & 1 & \ldots & -\gamma_m \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}. \tag{31}$$

4.2.3 Column replacement updating

Suppose we want to replace the pth column of a nonsingular matrix $\mathbf{B}$ with an arbitrary column vector (not necessarily a column in $\mathbf{B}$), say $\mathbf{a}_k = \{a_{ik}\}$. The updated matrix can then be written

$$\overline{\mathbf{B}} = \mathbf{B}(\mathbf{I} - \mathbf{e}_p\mathbf{e}_p^{\mathrm{T}}) + \mathbf{a}_k\mathbf{e}_p^{\mathrm{T}}, \tag{32}$$

or

$$\overline{\mathbf{B}} = \mathbf{B}(\mathbf{I} + (\boldsymbol{\alpha}_k - \mathbf{e}_p)\mathbf{e}_p^{\mathrm{T}}) = \mathbf{B}\mathbf{E}_p(\boldsymbol{\alpha}_k), \tag{33}$$

where we have introduced the "updated" column vector

$$\boldsymbol{\alpha}_k = \{\alpha_{ik}\} = \mathbf{B}^{-1}\mathbf{a}_k. \tag{34}$$

The inverse can now be obtained from (27) or (28) with $j = p$ and $\boldsymbol{\alpha} = \boldsymbol{\alpha}_k$:

$$\overline{\mathbf{B}}^{-1} = \begin{pmatrix} 1 & & -\alpha_{1k}/\alpha_{pk} & & \\ & \ddots & \vdots & & \\ & & 1/\alpha_{pk} & & \\ & & \vdots & \ddots & \\ & & -\alpha_{mk}/\alpha_{pk} & & 1 \end{pmatrix} \begin{matrix} \\ \\ p \\ \\ \\ \end{matrix} \mathbf{B}^{-1}. \tag{35}$$

This result will be used to update the basis-inverse in the simplex procedure.

4.2.4 Permutation matrices

Sometimes we need a precise way to express what happens when rows and/or columns in a matrix are rearranged. Let an arbitrary permutation

$$\pi = \begin{pmatrix} 1 & 2 & \dots & m \\ p_1 & p_2 & \dots & p_m \end{pmatrix} \tag{36}$$

be given. The result of applying π to the rows of a matrix $\mathbf{B}$ is another matrix

$$\mathbf{B}_{\mathrm{r}} = \mathbf{P}\mathbf{B}, \tag{37}$$

where the premultiplying factor is given by

$$\mathbf{P} = (\mathbf{e}_{p_1}, \dots, \mathbf{e}_{p_m}) \tag{38}$$

and is called a *permutation matrix*. If we instead apply π to the columns of $\mathbf{B}$ we would get

$$\mathbf{B}_{\mathrm{c}} = \mathbf{B}\mathbf{P}^{\mathrm{T}}, \tag{39}$$

still with $\mathbf{P}$ given by (38). We could also permute both the rows and the columns, using two permutation matrices $\mathbf{P}$ and $\mathbf{Q}$:

$$\mathbf{B}_{\mathrm{rc}} = \mathbf{P}\mathbf{B}\mathbf{Q}^{\mathrm{T}}. \tag{40}$$

In particular, the rows and columns can be symmetrically permuted:

$$\mathbf{B}_{\mathrm{sym}} = \mathbf{P}\mathbf{B}\mathbf{P}^{\mathrm{T}}. \tag{41}$$

Here the diagonal row of $\mathbf{B}$ is preserved in $\mathbf{B}_{\mathrm{sym}}$; only the order of its elements is altered.

Any permutation matrix is orthogonal,

$$\mathbf{P}\mathbf{P}^{\mathrm{T}} = \mathbf{P}^{\mathrm{T}}\mathbf{P} = \mathbf{I}. \tag{42}$$

4.2.5 LU-factorization

Any nonsingular matrix can be written as the product of a lower triangular factor $\mathbf{L}$ and an upper triangular factor $\mathbf{U}$, at least when we admit row (and/or column) interchanges. This fact is contained in

Proposition 2 (LU-theorem) *Given a matrix* $\mathbf{B} \in \mathsf{R}^{m\times m}$, *in which all the leading principal minors are nonsingular, then there exists a unique lower triangular matrix* $\mathbf{L} = \{\ell_{ij}\}$ *with* $\ell_{ii} = 1$ *and a unique upper triangular matrix* $\mathbf{U} = \{u_{ij}\}$, *so that*

$$\mathbf{B} = \mathbf{LU}, \tag{43}$$

or, written out in full,

$$\begin{pmatrix} b_{11} & \dots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{m1} & \dots & b_{mm} \end{pmatrix} = \begin{pmatrix} 1 & & \\ \vdots & \ddots & \\ \ell_{m1} & \dots & 1 \end{pmatrix} \begin{pmatrix} u_{11} & \dots & u_{1m} \\ & \ddots & \vdots \\ & & u_{mm} \end{pmatrix} \tag{44}$$

A proof of this theorem is given e.g. in Forsythe and Moler [10]. The necessary rearrangement of an arbitrary nonsingular matrix to qualify for the $\mathbf{LU}$ theorem can always be accomplished by shuffling its rows (i.e. premultiplying it by a permutation matrix), or by shuffling its columns, or both.

4.2.6 Decomposition of triangular matrices into elementary matrices

For triangular matrices we have a particularly simple factorization into elementary column matrices:

$$\mathbf{L} = \begin{pmatrix} 1 & & & & \\ \vdots & \ddots & & & \\ \ell_{k1} & \dots & 1 & & \\ \vdots & & \vdots & \ddots & \\ \ell_{m1} & \dots & \ell_{mk} & \dots & 1 \end{pmatrix} = \prod_{k=1}^{m} \begin{pmatrix} 1 & & & & \overset{k}{} \\ & \ddots & & & \\ & & 1 & & \\ & & \vdots & \ddots & \\ & & \ell_{mk} & & 1 \end{pmatrix} \tag{45}$$

and

$$\mathbf{U} = \begin{pmatrix} u_{11} & \dots & u_{1k} & \dots & u_{1m} \\ & \ddots & \vdots & & \vdots \\ & & u_{kk} & \dots & u_{km} \\ & & & \ddots & \vdots \\ & & & & u_{mm} \end{pmatrix} = \prod_{k=m}^{1} \begin{pmatrix} 1 & & \overset{k}{u_{1k}} & & \\ & \ddots & \vdots & & \\ & & u_{kk} & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}. \tag{46}$$

There is a similar pair of factorizations into elementary row matrices:

$$\mathbf{L} = \begin{pmatrix} 1 & & & & \\ \vdots & \ddots & & & \\ \ell_{k1} & \dots & 1 & & \\ \vdots & & \vdots & \ddots & \\ \ell_{m1} & \dots & \ell_{mk} & \dots & 1 \end{pmatrix} = \prod_{k=1}^{m} {}^{k} \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ \ell_{k1} & \dots & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix} \tag{47}$$

and

$$\mathbf{U} = \begin{pmatrix} u_{11} & \dots & u_{1k} & \dots & u_{1m} \\ & \ddots & \vdots & & \vdots \\ & & u_{kk} & \dots & u_{km} \\ & & & \ddots & \vdots \\ & & & & u_{mm} \end{pmatrix} = \prod_{k=m}^{1} {}^{k} \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & u_{kk} & \dots & u_{km} \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}. \tag{48}$$

The factor in (45) corresponding to $k = m$ is the unit matrix, and so is the factor in (47) for $k = 1$; these unit factors are displayed for reasons of symmetry.

For the LINPROG implementation of simplex we need only the forms (46) and (47). Corresponding expressions for the inverse matrices $\mathbf{U}^{-1}$ and $\mathbf{L}^{-1}$ are easily obtained by using (27) and (31). In particular this shows that the inverse of a lower triangular matrix is again lower triangular, and similarly for an upper triangular matrix.

4.3 Elementary product forms

It is an essential feature of large-scale linear programming that the basis matrix $\mathbf{B}$, or rather its inverse $\mathbf{B}^{-1}$, is expressed as a product of elementary matrices. Such a product form of $\mathbf{B}^{-1}$ is used and maintained in the simplex procedure; it is occasionally reconstructed in the process of re-inversion (Section 4.4). The advantage of the product representation is that we need store only those vectors that form the elementary matrices. These are typically sparse and may be held as packed arrays.

4.3.1 The standard product form

Let an arbitrary nonsingular matrix $\mathbf{B} \in \mathsf{R}^{m \times m}$ be given. It is instructive to describe the so-called *standard product form* of $\mathbf{B}$, in which each factor is an elementary column matrix, though this form is not directly applied in LINPROG.

To obtain the representation we consider $\mathbf{B} = \mathbf{B}_t$ the result of a t-step column replacement process beginning with the unit matrix $\mathbf{B}_0 = \mathbf{I}$. At step number k we transform the matrix $\mathbf{B}_{k-1}$ to $\mathbf{B}_k$ by replacing column p_k by some "entering" vector $\mathbf{a}_k$. This vector need not be one of the columns of $\mathbf{B}$ itself, as $\mathbf{a}_k$ may later on again be replaced by another column. In Section 4.2.3 we saw that such a column replacement corresponds to post-multiplication of the matrix $\mathbf{B}_{k-1}$ by an elementary column matrix:

$$\mathbf{B}_k = \mathbf{B}_{k-1}\mathbf{E}_{p_k}(\boldsymbol{\alpha}_k), \tag{49}$$

where

$$\boldsymbol{\alpha}_k = \mathbf{B}_{k-1}^{-1}\mathbf{a}_k. \tag{50}$$

Assuming that the process terminates after t steps with all the columns of $\mathbf{B}$ formed in right positions, we can write down the standard product form from (49):

$$\mathbf{B} = \prod_{k=1}^{t} \mathbf{E}_{p_k}(\boldsymbol{\alpha}_k). \tag{51}$$

More important for LP than $\mathbf{B}$ itself is $\mathbf{B}^{-1}$. The inverse of (51) reads

$$\mathbf{B}^{-1} = \prod_{k=t}^{1} \mathbf{E}_{p_k}(\boldsymbol{\eta}_k), \tag{52}$$

where

$$\boldsymbol{\eta}_k = \boldsymbol{\eta} = \mathbf{g}_{p_k}(\boldsymbol{\alpha}_k) \tag{53}$$

has the components stated in (29). Below we give a constructive description of the replacement process in algorithmic form using "pseudo-PASCAL":

$\mathbf{B}_0^{-1} = \mathbf{I}$;
FOR $k := 1$ **TO** t **DO**
BEGIN
(* identify pivot index for this step *)

$j := p_k$;
(* replace current column j with some vector $\mathbf{a}_k$*)
$\alpha_k := \mathbf{B}_{k-1}^{-1}\mathbf{a}_k$;
$\eta_k := \mathbf{g}_j(\alpha_k)$;
$\mathbf{B}_k^{-1} := \mathbf{E}_j(\eta_k)\mathbf{B}_{k-1}^{-1}$
END;
$\mathbf{B}^{-1} := \mathbf{B}_t^{-1}$;

It is clear that many different replacement sequences may lead from $\mathbf{I}$ to the same matrix $\mathbf{B}$ or $\mathbf{B}^{-1}$. Consequently the representations (51) and (52) are not unique. Nor is the number t of exchange steps. For example, at the end of a re-inversion process we may let $t = m$. Subsequently the simplex optimization process may add new factors to the product, and we may get $t > m$.

4.3.2 Product form of the LU type

The standard product form described here has been in widespread use in earlier LP codes. Its merit is its simplicity. Modern implementations of simplex, however, use a product form which is still made up of elementary matrices, but is related to a triangular factorization of the basis matrix. This complication in structure is more than offset by savings in storage and calculations.

Immediately after a re-inversion (and at the start of the simplex process) we have an $\mathbf{LU}$-factorization (43) – (44) of the basis matrix $\mathbf{B}$. Among the different ways of decomposing $\mathbf{U}$ and $\mathbf{L}$ into products of elementary matrices we select those given by (46) and (47), because this choice gives us a product form compatible with the Forrest-Tomlin updating scheme given in Section 4.5. Hence we write

$$\mathbf{L} = \mathbf{L}_1\mathbf{L}_2 \ldots \mathbf{L}_k \ldots \mathbf{L}_m, \tag{54}$$

where

$$\mathbf{L}_k = \begin{matrix} \\ \\ k \\ \\ \\ \end{matrix} \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ \ell_{k1} & \ldots & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}, \tag{55}$$

and

$$\mathbf{U} = \mathbf{U}_m\mathbf{U}_{m-1} \ldots \mathbf{U}_k \ldots \mathbf{U}_1, \tag{56}$$

where

$$\mathbf{U}_k = \begin{matrix} \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ k \\ \\ \\ \end{matrix} \begin{pmatrix} 1 & & u_{1k} & & \\ & \ddots & \vdots & & \\ & & u_{kk} & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}. \tag{57}$$

The resulting basis-inverse becomes

$$\mathbf{B}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1} = \mathbf{U}_1^{-1} \ldots \mathbf{U}_k^{-1} \ldots \mathbf{U}_m^{-1}\mathbf{L}_m^{-1} \ldots \mathbf{L}_k^{-1} \ldots \mathbf{L}_1^{-1}, \tag{58}$$

where

$$\mathbf{U}_k^{-1} = \begin{pmatrix} 1 & & -u_{1k}/u_{kk} & & \\ & \ddots & \vdots & & \\ & & 1/u_{kk} & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix} \quad \begin{matrix} \\ \\ k \\ \\ \\ \end{matrix} \tag{59}$$

and

$$\mathbf{L}_k^{-1} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ -\ell_{k1} & \dots & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix} \quad \begin{matrix} \\ \\ k \\ \\ \\ \end{matrix} . \tag{60}$$

These expressions illustrate the close relationship between **LU** decomposition and matrix inversion in product form.

When the simplex procedure calls for a column exchange in the basis, we must update the representation (58) accordingly. As we shall see in Section 4.5 the Forrest-Tomlin procedure maintains $\mathbf{B}^{-1}$ in a more general form than (58): After each simplex step it leaves the basis-inverse as a product

$$\mathbf{B}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1} = \mathbf{U}_1^{-1} \dots \mathbf{U}_i^{-1} \dots \mathbf{U}_m^{-1}\mathbf{L}_l^{-1} \dots \mathbf{L}_s^{-1} \dots \mathbf{L}_1^{-1}, \tag{61}$$

where $\mathbf{U} = \mathbf{U}_m \dots \mathbf{U}_1$ and $\mathbf{L} = \mathbf{L}_1 \dots \mathbf{L}_l$. The factors $\mathbf{U}_i$ (and $\mathbf{U}_i^{-1}$) are elementary column matrices, while $\mathbf{L}_s$ (and $\mathbf{L}_s^{-1}$) are elementary row matrices. But $\mathbf{U}_i$ and $\mathbf{L}_s$ need no longer be triangular, nor need $\mathbf{L}$ and $\mathbf{U}$ in (61). Again, the factorization (61) is not unique, and t, the number of factors $\mathbf{L}_s$, may be greater than m.

In LINPROG we store the factors $\mathbf{U}_i^{-1}$ in (61) as a sequence of column vectors in one packed list. This "eta-U" list is an array in the computer (which we assume has virtual memory), but for traditional reasons we also call it the "U file". This file (together with a list of the pivot indices) determines the factor $\mathbf{U}^{-1}$. We shall also maintain an "eta-L" list or "L file" containing a sequence of row vectors forming the elementary row matrices $\mathbf{L}_s^{-1}$. It determines the other factor $\mathbf{L}^{-1}$. Taken together, the L and U files form the so-called "eta-file"; we say that this file contains "column-eta" vectors and "row-eta" vectors.

4.4 Re-inversions

We have seen that $\mathbf{B}^{-1}$, the inverse of the basis matrix $\mathbf{B}$, in LINPROG is represented by two so-called "eta lists" associated with the two factors of the **LU** product form described in the previous section. For each simplex iteration the lists are augmented with new elementary vectors (to be discussed in Section 4.5), and this growth causes the time per iteration as well as the roundoff errors to grow. Eventually it becomes necessary to compress the structure, and we then make a fresh inversion of $\mathbf{B}$, using our knowledge of the present set of basic columns. Different heuristic criteria for evoking an inversion exist. One popular way is to let the CPU clock of the computer trigger the inversion. Also the detection of unexpected infeasibilities, or numerical deterioration of the current solution, could release an inversion. In LINPROG the re-inversions are performed at regular intervals of the simplex iteration counter, cf. the parameter MITRE described in Section 3.2.

In the following we present the various ingredients of the inversion procedure. After this we state the total inversion algorithm in compressed form, and finally we discuss some possible ways of improving the implementation scheme in LINPROG.

4.4.1 Decomposition arithmetics and eta vectors

The output from the inversion process in LINPROG will be a product representation (58) for $\mathbf{B}^{-1}$, which in turn is based on the triangular **LU** factorization (43) – (44) of the basis matrix **B**; for this reason the name "re-factorization" is sometimes used instead of re-inversion [4]. We assume that the rows and columns of **B** are already permuted in a way that makes the representation (44) possible; the choice of such permutations is the topic of a later discussion.

The matrix identity (44) for the **LU** decomposition determines the entries ℓ_{ij} and u_{ij} of **L** and **U**. In fact (44) can be written as the single scalar equation

$$\sum_{\nu=1}^{\min(i,j)} \ell_{i\nu} u_{\nu j} = b_{ij}; \quad i = 1, \ldots, m; \; j = 1, \ldots, m, \tag{62}$$

together with

$$\ell_{ii} = 1. \tag{63}$$

There are several possible orders in which the elements can be computed. We have chosen an order that matches the construction of the row vector in $\mathbf{L}_k$ of (55) and the column vector in $\mathbf{U}_k$ of (57); the calculations are made clear from the following piece of pseudo-PASCAL code:

FOR $k := 1$ **TO** m **DO**
  BEGIN
    FOR $j := 1$ **TO** $k - 1$ **DO**

$$\ell_{kj} := (b_{kj} - \sum_{\nu=1}^{j-1} \ell_{k\nu} u_{\nu j})/u_{jj}; \tag{64}$$

    FOR $i := 1$ **TO** k **DO**

$$u_{ik} := b_{ik} - \sum_{\nu=1}^{i-1} \ell_{i\nu} u_{\nu k} \tag{65}$$

  END;

We see that the loop (64) containing ℓ_{kj} produces the row vector in (55), and the loop (65) containing u_{ik} produces the column vector in (57). We also observe that the last element computed at stage k is u_{kk}, which we call the pivot element. Our decomposition algorithm may be characterized as a non-standard variant of Crout's method.

The row eta vector actually stored by LINPROG at pivot stage k is the left-diagonal part $(-\ell_{k1}, \ldots, -\ell_{k,k-1})$ of row k in $\mathbf{L}_k^{-1}$, equation (60); as the column eta we store the super-diagonal part $(u_{1k}, \ldots, u_{k-1,k})^{\mathrm{T}}$ of column k of $\mathbf{U}_k$, equation (57), while the pivot element u_{kk} is stored in a separate list. In this way we avoid the explicit division by the pivot, and yet have all the information needed for (59).

4.4.2 Pre-ordering of rows and columns

In the previous algebraic description of the inversion process we assumed that **B** was already organized in a way that permits us to pivot down the diagonal, as implied by the decomposition (44). In this connection it is important to realize that what is essential for a basis matrix **B** is its *collection* of independent columns, not the internal ordering of these columns within **B**. Thus, when the inversion is due, we are given a set of basic columns of the constraint matrix **A** in (5), and we are free to arrange them in any order. The rows of **B** correspond to rows in **A** and

can also be re-shuffled; this means that we mention the constraints in a different order. Hence we may arrange both the columns and rows in any way that will suit us; only we must keep track of the necessary permutations by index arrays pointing to the row and column positions, respectively, of the matrix **A**. The final row and column numbering of **B** is simply defined by the order of the pivot assignments of the same rows and columns. With this "chronological" convention for the numbering, the output from the inversion will be an unpermuted triangular factorization (44). Later, when the simplex procedure exchanges one column with another from **A**, the new basic column simply inherits its number from the old one. We shall try to utilize the freedom we have to pre-order the basis matrix in such a way that the sparsity is preserved as well as possible without sacrificing the numerical stability. It is well-known that although we start with a sparse basis **B**, the **LU** decomposition may cause "fill-in" of new nonzero elements (the opposite process, viz. cancellation, is possible though less likely.) A measure of the total fill-in is the excess of nonzeros in the combined matrix formed by **L** and **U** over the nonzeros in **B** itself. This fill-in depends in a critical way on the pivot selection order. If we can reduce it, we get smaller eta lists after the inversion. This in turn means faster subsequent simplex iterations.

4.4.3 Pivot selection strategy for sparsity preservation

If it were possible to arrange **B** in a strictly lower-triangular form, the **LU** factorization would be trivial, and we would get no fill-in. This ideal goal is seldom attained in practice. Our selection method resembles that described by Orchard-Hays [11], who builds a partly triangular basis with a square block **T** in the middle (see Figure 1). The submatrix **T** is called the *nucleus* (or "bump"). In particular

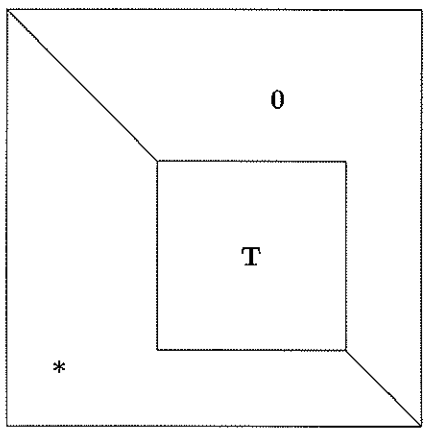


Figure 1. Pre-ordering of **B** *into a block triangular form with a nucleus*

T may be null, or it may be equal to **B** itself. The upper-left part of the figure represents "singleton rows", while the lower-right part represents "singleton columns". These rows and columns can easily be identified successively. Creation of new nonzeros depends on the sparsity pattern of **B** and is governed by (64) and (65); we see from these equations that the fill-in is limited to within and below the nucleus.

But we can do still better than this. If we pivot literately in the sequence implied by Figure 1, we would have to set the singleton columns aside, until the nucleus was processed. This would require extra workspace. We avoid it by permuting the

rows and columns in **B** such that the rows below the nucleus are moved to the top (in reverse order) with analogous shifts of the columns (cf., Benichou *et al.*, [12]). One can show that this gives the arrangement in Figure 2, and thus leads to a sequence where we first process the singleton columns, then the singleton rows,

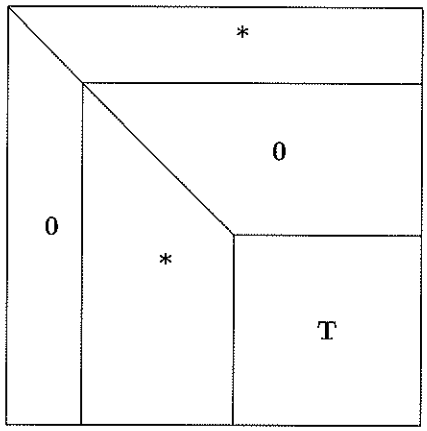


Figure 2. Rearranged pre-ordering with the nucleus at the end

and finally the nucleus. An additional gain with Figure 2 is that there is no fill-in outside the nucleus itself.

When we reach the nucleus we must use some pivot selection heuristics. Let us in the first place concentrate on limitation of fill-in, hence preservation of sparsity. A reasonable strategy will be to select first the new pivot row as one with a minimum number of active nonzeros, and then choose the corresponding pivot column as one with a nonzero in the pivot row and with a minimum count of nonzeros in the column. "Active" means here: "not yet assigned for pivoting". We notice that with this rule for the nucleus there is no need for a separate treatment of the singleton rows; they will automatically be selected as the first rows.

When pivots are assigned, the corresponding row and column are de-activated, and appropriate updating of lists take place.

4.4.4 Modifications to ensure numerical stability

The pivot selection rule described above works exclusively on the sparsity pattern of the basis. As it stands, it does not involve numerical considerations. But it is quite possible that the computed value u_{kk} of the proposed pivot element becomes exactly zero, or at least its absolute value $|u_{kk}|$ becomes very small, and in both cases the pivot must be rejected.

LINPROG uses the following amendment of the selection rule given in Section 4.4.3: The pivot row which was selected from sparsity considerations is always accepted. When selecting a pivot column, we find two candidates to choose between (they may coincide). The first one is selected by the minimum-count criterion given in Section 4.4.3; ties are resolved in favour of greater pivot size. The second one has maximum pivot size among all the available columns. Denoting the pivot values for the two candidates p_1 and p_2, respectively, we use the former candidate if $|p_1/p_2|$ exceeds some number ε, otherwise the latter one. If we take $\varepsilon = 1$, we would always select p_2, and this corresponds to the classical (partial) pivoting rule for solving linear equations. On the other hand, a small ε would favour the choice of p_1, i.e. the pivoting for sparsity and limitation of the fill-in. LINPROG uses

$\varepsilon = 0.01$ as a standard value. It can be changed to another value by the user with the keyword EPSRIN in the Control File, cf. Section 3.2 and Table 3 in Section 4.6.5.

The described acceptance test is based on the relative size of the pivot to the maximum size of all available pivots in the row. Hence we must calculate all these pivots. Instead we could have used a cheaper absolute test and accept the pivot that was proposed by the sparsity rule, unless its size drops below an absolute threshold ε_{thr}; in that case one would invoke an "emergency procedure" and select the largest possible pivot size. Our numerical experiments indicate that the relative test often gives better stability than does the absolute test, and that the overall increase in computing time is small. Some other inversion schemes use a combination of both tests.

Recall that the computation of a pivot element is made by (65) with $i = k$, while (65) for $i < k$ and (64) give us the new column and row eta vectors, respectively.

4.4.5 Sparse-matrix implementation details

The "Boolean" part of the inversion requires that we maintain and update a map of the current pattern of nonzeros during our **LU** decomposition. Zeros as a result of cancellation are ignored. In our FORTRAN code we use traditional sparse-matrix techniques, based on a double set of ordered lists: one list, IB, is column-ordered and contains the row numbers of the nonzeros, while the other one, JB, is row-ordered and contains the column numbers of the same elements. In both lists we reserve "elbow room" to accommodate for possible fill-in (initially we provide the same elbow room for each row/column as its count of nonzeros.) Each of the two lists are equipped with three set of pointer arrays. For the column-ordered list they are ICLPT1, ICLPT2, and ICLAST. ICLPT1(J) holds the address in IB that corresponds to the first non-zero in column J. ICLPT2(J) points in the same way to the last nonzero of the column. ICLAST(J) holds the last free elbow-room address for column J. Analogous arrays IRWPT1, IRWPT2, and IRLAST support the row-ordered list.

Should the elbow room for some column or row be used up during the inversion, a simple memory-management procedure is activated: we make a fresh copy of the column or row at the end of the list with a new elbow room equal to the new information length. The previous storage for that column or row is wasted, and no garbage collection is made. This waste may be tolerated, because after the inversion is finished, all the lists are abandoned.

Moreover we must keep track of the still active counts of nonzeros in the rows and in the columns. These counts are used in our pre-ordering method for pivot selection. In the latest version of LINPROG a double linked list, i.e. a list with forward and backward pointers, ensures a quick retrieval of the pivot rows in the nucleus according to the minimal-count criterion. Another linked list is used for the selection of singleton columns. These improvements, which are due to Antti Lehtilä and Pekka Pirilä from VTT in Finland, give a significant reduction in the CPU time for large problems.

Special care must be taken when computing the scalar products entering (64) and (65). The problem arising is that the **L** and **U** elements, which are loaded from the two eta files, correspond to unpermuted rows and columns of the constraint matrix. But we know that both should be permuted to match our common pivot-index numbering, and this makes an internal sorting procedure necessary.

4.4.6 Algorithmic description of the inversion

After this discussion of the different aspects of the inversion method, we shall now put the bricks together and give a compressed algorithmic description (expressed in pseudo-PASCAL). We assume that all the necessary initializations are made. Then we proceed as follows:

```
clsing := TRUE;
(* clsing becomes FALSE when singleton columns are exhausted *)
FOR k := 1 TO m DO
(* k is the pivot step counter *)
BEGIN
  200:
  IF clsing THEN
  BEGIN
    Locate a singleton pivotcolumn (using a linked list);
    clsing := (we found a singleton column);
    IF ¬clsing THEN GO TO 200;
    Locate corresponding pivot row;
  END ELSE
  BEGIN (* code for nucleus including singleton rows *)
    Locate pivotrow as one with minimum count
    by using a double linked list;
  END (* nucleus *);
  (* selection of pivotrow finished *)
  Build new row eta vector by (64);
  De-activate pivotrow and give it the "chronological" number k;
  IF ¬clsing THEN
  BEGIN (* code for nucleus including singleton rows *)
    Scan pivotrow to produce pivotcolumn candidates 1 and 2;
    (*
    candidate 1 has minimum count (ties broken in favour of pivot size);
    candidate 2 has largest pivot size among all available columns;
    the pivot values are p1 and p2, respectively
    *)
    IF |p1| > ε|p2| THEN pivotcolumn := candidate1
    ELSE pivotcolumn := candidate2
  END (* nucleus *);
  Build new column eta vector by (65);
  De-activate pivotcolumn and give it the "chronological" number k;
  Update basic column indicator;
  Reduce active row and column counts with the de-activated elements;
  Update ordered lists to accommodate fill-in at current pivot step;
  IF no more elbow-room for a column or row THEN
  BEGIN
    (* perform memory management *)
    Copy row or column to end of ordered list;
    Provide fresh elbow room of size equal to copied part
  END
END;
```

4.4.7 Possible improvements

It is generally agreed that the **LU** decomposition described here is superior to the old "Product Form of the Inverse" (PFI) method, in which $\mathbf{B}^{-1}$ is factored in standard product form using elementary column matrices only, as described in Section 4.3.1. The inversion time is less, and the resulting eta lists are shorter.

On the other hand, there is still room for improvements of the inversion, not yet exploited in LINPROG. It should be possible to reduce the fill-in by more sophisticated methods of pivot selection. First, one could carry out a complete block triangular rearrangement of **B**. There exist efficient algorithms to do this (see Duff and Reid [13], and Duff [14]). Another way to pursue, proposed by Hellerman and Rarick [15], would be to look at **B** (or each of the triangularization blocks in turn), and by suitable re-shuffling of the rows and columns transform it to a triangular structure superimposed by "spikes". The advantage is that all the fill-in is confined to the spikes, and this tends to reduce the overall inversion work. Both block triangularization and spike techniques are common ingredients of today's commercial simplex codes.

Finally, it might be questioned whether our ordered-list representation of the sparsity pattern, with its overhead of elbow room and memory management, could stand up against an implementation based on linked lists.

4.5 The Forrest-Tomlin procedure

In this section we turn to the question of updating the factorization of the basis matrix **B**, or its inverse $\mathbf{B}^{-1}$, when successive iteration cycles of the simplex process cause replacements of the columns of **B**. We shall in the following describe the *Forrest-Tomlin* updating method [16, 17, 8, 4], which is suitable for maintaining an **LU**-like product form (61) for $\mathbf{B}^{-1}$.

4.5.1 General outline

In Section 4.4 it was pointed out, that when we resume the simplex iterations after a re-inversion, our current basis matrix $\mathbf{B} \in \mathsf{R}^{m \times m}$ will be factored as

$$\mathbf{B} = \mathbf{L}\mathbf{U}, \tag{66}$$

where **L** is lower triangular and **U** upper triangular, provided we adopt the row and column numbering induced by the pivoting sequence of the inversion. We shall do so, adhering to the numbering convention given in Section 4.4.2.

When a subsequent iteration step of the simplex process transforms **B** to an adjacent basis matrix $\overline{\mathbf{B}}$, the question arises whether it is still possible to write $\overline{\mathbf{B}}$ as a product of factors $\overline{\mathbf{L}}$ and $\overline{\mathbf{U}}$,

$$\overline{\mathbf{B}} = \overline{\mathbf{L}}\,\overline{\mathbf{U}}, \tag{67}$$

such that $\overline{\mathbf{L}}$ can be easily computed by updating **L**, and analogously for $\overline{\mathbf{U}}$. Several schemes exist that accomplish this task. The merit of the Forrest-Tomlin method is that it preserves the triangular factorization during updating, when allowance for permutations is made. Moreover, it creates no fill-in of new nonzeros in the eta lists.

In order to give a general description of the updating mechanism, let us temporarily leave any specific assumptions about **L** and **U** in (66) out of account. For the time being they are just invertible matrices whose product is **B**. Suppose now that we want to carry out an exchange step of the simplex procedure with pivot index p. This means that we must replace column p of **B** by some nonbasic column $\mathbf{a}_k$ of the constraint matrix **A**. We use the ordering (7) for the columns

in $\mathbf{A}$ and assume that the qth nonbasic column was chosen such that $\mathbf{a}_k = \mathbf{a}_{m+q}$. The resulting adjacent basis matrix $\overline{\mathbf{B}}$ is given by (32) with $k = m + q$, and this we shall factorize in the form (67), or in the equivalent inverse form

$$\overline{\mathbf{B}}^{-1} = \overline{\mathbf{U}}^{-1}\overline{\mathbf{L}}^{-1}. \tag{68}$$

We introduce the "partially updated" incoming vector

$$\mathbf{v} = \mathbf{L}^{-1}\mathbf{a}_{m+q}; \tag{69}$$

if we replace column p of $\mathbf{U}$ by $\mathbf{v}$ and again apply the column replacement formula (32) we obtain the matrix

$$\mathbf{U}' = \mathbf{U}(\mathbf{I} - \mathbf{e}_p\mathbf{e}_p^{\mathrm{T}}) + \mathbf{v}\mathbf{e}_p^{\mathrm{T}}, \tag{70}$$

and evidently we have

$$\overline{\mathbf{B}} = \mathbf{L}\mathbf{U}'. \tag{71}$$

Next we define the vector $\mathbf{r}$ as the unique solution to the equation

$$\mathbf{U}^{\mathrm{T}}\mathbf{r} = \mathbf{e}_p, \tag{72}$$

from which we derive

$$\mathbf{r}^{\mathrm{T}} = \mathbf{e}_p^{\mathrm{T}}\mathbf{U}^{-1}, \tag{73}$$

which states that $\mathbf{r}^{\mathrm{T}}$ equals the pth row of $\mathbf{U}^{-1}$. Now build the elementary row matrix

$$\mathbf{R} = \mathbf{I} - \mathbf{e}_p\mathbf{e}_p^{\mathrm{T}} + \sigma\mathbf{e}_p\mathbf{r}^{\mathrm{T}} = \begin{matrix} \\ \\ p \\ \\ \\ \end{matrix}\begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ \cdots & \cdots & \sigma r^{\mathrm{T}} & \cdots & \cdots \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}, \tag{74}$$

where the normalization constant σ is chosen such as to render the pivot element of $\mathbf{R}$ equal to one, that is

$$\sigma = 1/r_p. \tag{75}$$

Then define

$$\overline{\mathbf{L}} = \mathbf{L}\mathbf{R}^{-1} \tag{76}$$

and

$$\overline{\mathbf{U}} = \mathbf{R}\mathbf{U}'. \tag{77}$$

These are the updated factors in (67) we are looking for, and (71) shows that their product is indeed $\overline{\mathbf{B}}$.

What can be said about the structure of the updated factors $\overline{\mathbf{L}}$ and $\overline{\mathbf{U}}$ and their inverse? If we first consider $\overline{\mathbf{L}}$, we observe that immediately after an inversion $\mathbf{L}$ is lower triangular and hence can be factored into a product of elementary row matrices, as (47) shows. As $\mathbf{R}^{-1}$ is also an elementary row matrix, the expression (76) shows that the updating process preserves $\overline{\mathbf{L}}$ as a product of elementary row matrices. Of course the same is true for the inverse

$$\overline{\mathbf{L}}^{-1} = \mathbf{R}\mathbf{L}^{-1}. \tag{78}$$

The structure of $\overline{\mathbf{U}}$ can be deduced from (77), (70), (74), and (73). After reduction we find

$$\overline{\mathbf{U}} = (\mathbf{I} - \mathbf{e}_p\mathbf{e}_p^{\mathrm{T}})\mathbf{U}(\mathbf{I} - \mathbf{e}_p\mathbf{e}_p^{\mathrm{T}}) + \mathbf{v}'\mathbf{e}_p^{\mathrm{T}}, \tag{79}$$

where we have introduced the vector

$$\mathbf{v}' = \mathbf{R}\mathbf{v}; \tag{80}$$

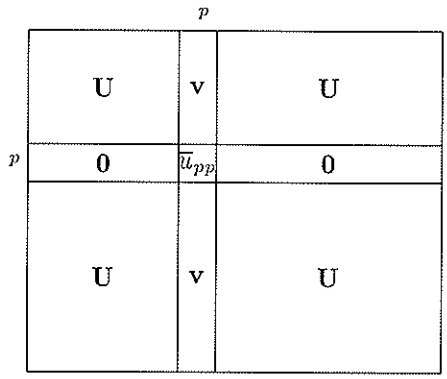


Figure 3. General structure of the matrix $\overline{\mathbf{U}}$

we see that $\mathbf{v}'$ equals $\mathbf{v}$ in all places except the pth where its element is

$$v'_p = \overline{u}_{pp} = \sigma \; \mathbf{r}^{\mathrm{T}}\mathbf{v}. \tag{81}$$

A sketch of $\overline{\mathbf{U}}$ is given in Figure 3. It is seen that the effect of $\mathbf{R}$ is to annihilate all nonpivotal elements of the pivot row of $\mathbf{U}'$.

Let us now assume that $\mathbf{U}$ is upper triangular (as it is after an inversion). Then $\overline{\mathbf{U}}$ takes the form shown in Figure 4. The cyclic permutation $(m, m-1, \ldots, p+1, p)$

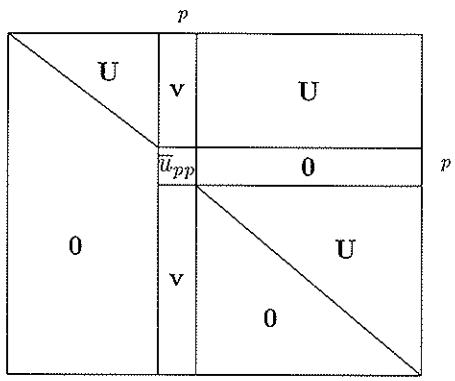


Figure 4. Almost-triangular structure of the matrix $\overline{\mathbf{U}}$

has the effect of taking element number p to position m and shifting the elements $p+1, \ldots, m$ one place to the left. If $\mathbf{Q}$ is the corresponding permutation matrix (Section 4.2.4), then the symmetrically permuted matrix

$$\mathbf{U}^* = \mathbf{Q}\overline{\mathbf{U}}\mathbf{Q}^{\mathrm{T}} \tag{82}$$

will be upper triangular (and because of (73) $\mathbf{R}$ will be upper triangular, too).

More generally, suppose that $\mathbf{U}$ is a SPUT matrix (SPUT = Symmetrically Permuted Upper Triangular). Then one infers that the updated matrix $\overline{\mathbf{U}}$ of Figure 3 is also a SPUT matrix: we need one symmetric permutation to transform Figure 3 to a matrix with the same structure as that in Figure 4, and another one to render this upper triangular. As the composition of two symmetric permutations is again symmetric, this argument shows that the updating process preserves the SPUT property of $\overline{\mathbf{U}}$ also after successive simplex exchange steps.

4.5.2 Updating of product representation

Next we shall describe how the Forrest-Tomlin process maintains the representation of $\mathbf{B}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}$ as a product of elementary row and column matrices. We will use an inductive argument to show that the current representation of the basis-inverse will be given by (61), where m is the order of $\mathbf{B}$ and t is a pivot step counter, which starts from the value m after a re-inversion and increases by 1 for each subsequent simplex iteration. When $t = m$, the expression (61) coincides with the inversion formula (58) as it should. The L-part of formula (61),

$$\mathbf{L}^{-1} = \mathbf{L}_t^{-1} \ldots \mathbf{L}_s^{-1} \ldots \mathbf{L}_1^{-1}, \tag{83}$$

clearly follows from the discussion in Section 4.5.1; the new factor $\mathbf{L}_t^{-1}$ to be added at the current step t equals the elementary row matrix $\mathbf{R}$ in (74); hence it can be written in the form

$$\mathbf{L}_t^{-1} = p \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ \ell_{p1} & \ldots & 1 & \ldots & \ell_{pm} \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}, \tag{84}$$

p being the new pivot index. We note that for the pivot element in (84) we have

$$\ell_{pp} = 1. \tag{85}$$

The U-part of formula (61) is

$$\mathbf{U}^{-1} = \mathbf{U}_1^{-1} \ldots \mathbf{U}_i^{-1} \ldots \mathbf{U}_m^{-1}. \tag{86}$$

Notice the asymmetry between (83) and (86) regarding their number of factors. As we shall presently see, this peculiarity is related to the fact that the Forrest-Tomlin process leaves the number m of elementary factors in the U-inverse fixed.

To examine the U-part of the updating process in more detail, suppose that the previous re-inversion and subsequent simplex steps have resulted in a SPUT matrix $\mathbf{U}$. Then there exists a truly upper triangular matrix

$$\mathbf{U}^* = \begin{pmatrix} u_{11}^* & u_{12}^* & \ldots & u_{1m}^* \\ & u_{22}^* & \ldots & u_{2m}^* \\ & & \ddots & \vdots \\ & & & u_{mm}^* \end{pmatrix} \tag{87}$$

and a permutation (36) which transforms $\mathbf{U}^*$ into $\mathbf{U}$ when applied to the rows as well as the columns. We may express this fact in terms of the corresponding permutation matrix (38) by the equivalent relations

$$\mathbf{U} = \mathbf{P}\mathbf{U}^*\mathbf{P}^{\mathrm{T}} \tag{88}$$

and

$$\mathbf{U}^{-1} = \mathbf{P}(\mathbf{U}^*)^{-1}\mathbf{P}^{\mathrm{T}}, \tag{89}$$

where we use the orthogonality property (42) of $\mathbf{P}$. Referring again to the permutation (36), the relation (88) shows that the element in the p_ith place in the diagonal row of $\mathbf{U}$ is identical with u_{ii}^*. For this reason p_i is called the ith pivot index and u_{ii}^* the ith pivot element.

By formula (46) $\mathbf{U}^*$ can be written as a product of elementary column matrices

$$\mathbf{U}^* = \mathbf{U}_m^* \ldots \mathbf{U}_i^* \ldots \mathbf{U}_1^*, \tag{90}$$

where

$$\mathbf{U}_i^* = \begin{pmatrix} 1 & & u_{1i}^* & & \\ & \ddots & \vdots & & \\ & & u_{ii}^* & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ i \\ \\ \\ \end{matrix} \tag{91}$$

is upper triangular. By (88) we obtain a similar product representation for $\mathbf{U}$:

$$\begin{aligned} \mathbf{U} &= \mathbf{P}\mathbf{U}_m^* \ldots \mathbf{U}_i^* \ldots \mathbf{U}_1^* \mathbf{P}^{\mathrm{T}} \\ &= (\mathbf{P}\mathbf{U}_m^* \mathbf{P}^{\mathrm{T}}) \ldots (\mathbf{P}\mathbf{U}_i^* \mathbf{P}^{\mathrm{T}}) \ldots (\mathbf{P}\mathbf{U}_1^* \mathbf{P}^{\mathrm{T}}) \\ &= \mathbf{U}_m \ldots \mathbf{U}_i \ldots \mathbf{U}_1, \end{aligned} \tag{92}$$

where we again have used (42). Each factor in (92), defined as

$$\mathbf{U}_i = \mathbf{P}\mathbf{U}_i^* \mathbf{P}^{\mathrm{T}}, \tag{93}$$

is an elementary column matrix of the form (cf. (25))

$$\mathbf{U}_i = \begin{pmatrix} 1 & & \cdot & & \\ & \ddots & \vdots & & \\ & & u_{ii}^* & & \\ & & \vdots & \ddots & \\ & & \cdot & & 1 \end{pmatrix} \begin{matrix} \\ \\ p_i \\ \\ \\ \end{matrix} = \mathbf{E}_{p_i}(\mathbf{y}), \tag{94}$$

p_i being the ith pivot index; the column vector $\mathbf{y} = \{y_\ell\}$ has the same elements as column i in (91), only permuted:

$$y_{p_\ell} = u_{\ell i}^*, \qquad \ell = 1, \ldots, m, \tag{95}$$

where, of course, $u_{\ell i}^* = 0$ for $\ell > i$. The inverse of (92) reads

$$\mathbf{U}^{-1} = \mathbf{U}_1^{-1} \ldots \mathbf{U}_i^{-1} \ldots \mathbf{U}_m^{-1}, \tag{96}$$

where, as a consequence of (27) – (29), $\mathbf{U}_i^{-1}$ is again an elementary column matrix,

$$\mathbf{U}_i^{-1} = \begin{pmatrix} 1 & & \cdot & & \\ & \ddots & \vdots & & \\ & & 1/u_{ii}^* & & \\ & & \vdots & \ddots & \\ & & \cdot & & 1 \end{pmatrix} \begin{matrix} \\ \\ p_i \\ \\ \\ \end{matrix} = \mathbf{E}_{p_i}(\mathbf{z}), \tag{97}$$

with the column vector $\mathbf{z}$ defined by

$$z_{p_\ell} = -\frac{y_{p_\ell}}{y_{p_i}} = -\frac{u_{\ell i}^*}{u_{ii}^*}, \qquad \ell = 1, \ldots, m, \quad \ell \neq i \tag{98}$$

and

$$z_{p_i} = \frac{1}{u_{ii}^*}. \tag{99}$$

To perform the updating we first compute $\mathbf{r}^{\mathrm{T}}$ from (73), which in product form reads

$$\mathbf{r}^{\mathrm{T}} = \mathbf{e}_p^{\mathrm{T}} \mathbf{U}_1^{-1} \ldots \mathbf{U}_m^{-1}. \tag{100}$$

This gives us the elementary row matrix $\mathbf{R}$ defined by (74). The new pivot index p is equal to the pivot index p_i for one of the factors $\mathbf{U}_i$ given by (94), say for

$i = k$, so we have $p = p_k$; we call $\mathbf{U}_k$ the "pivotal" factor. We can show that we are entitled to discard the "pre-pivotal" factors in (100) and write

$$\mathbf{r}^{\mathrm{T}} = \mathbf{e}_p^{\mathrm{T}} \mathbf{U}_k^{-1} \dots \mathbf{U}_m^{-1}. \tag{101}$$

To see this we insert (89) in (73),

$$\mathbf{r}^{\mathrm{T}} = \mathbf{e}_p^{\mathrm{T}} \mathbf{P} (\mathbf{U}^*)^{-1} \mathbf{P}^{\mathrm{T}}, \tag{102}$$

take the inverse of (90),

$$(\mathbf{U}^*)^{-1} = (\mathbf{U}_1^*)^{-1} \dots (\mathbf{U}_m^*)^{-1}, \tag{103}$$

and use

$$\mathbf{e}_p^{\mathrm{T}} \mathbf{P} = \mathbf{e}_k^{\mathrm{T}}. \tag{104}$$

Then we get

$$\mathbf{r}^{\mathrm{T}} = \mathbf{e}_k^{\mathrm{T}} (\mathbf{U}_1^*)^{-1} \dots (\mathbf{U}_k^*)^{-1} \dots (\mathbf{U}_m^*)^{-1} \mathbf{P}^{\mathrm{T}}, \tag{105}$$

and when we follow $\mathbf{e}_k^{\mathrm{T}}$ through the successive multiplications, the first $k-1$ of these are neutral operations, hence

$$\mathbf{r}^{\mathrm{T}} = \mathbf{e}_k^{\mathrm{T}} (\mathbf{U}_k^*)^{-1} \dots (\mathbf{U}_m^*)^{-1} \mathbf{P}^{\mathrm{T}}. \tag{106}$$

Finally, when we apply (93), (104), and (42), we get (101).

If we omit the pivotal factor from (101), (106) shows that the only effect is to multiply $\mathbf{r}^{\mathrm{T}}$ by a scalar. It is easy to see that the pth element in the new product becomes 1, hence we have

$$\sigma \mathbf{r}^{\mathrm{T}} = \mathbf{e}_p^{\mathrm{T}} \mathbf{U}_{k+1}^{-1} \dots \mathbf{U}_m^{-1}; \tag{107}$$

this formula is useful because it is $\sigma \mathbf{r}^{\mathrm{T}}$ rather than $\mathbf{r}^{\mathrm{T}}$ which enters in the updating process, as (74) and (81) indicate.

Next we shall show that the transformed matrix $\overline{\mathbf{U}}$ defined in (79) and depicted in Figure 3, like $\mathbf{U}$ in (92), can be written as a product of m elementary column matrices,

$$\overline{\mathbf{U}} = \mathbf{C} \tilde{\mathbf{U}}_m \dots \tilde{\mathbf{U}}_{k+1} \mathbf{U}_{k-1} \dots \mathbf{U}_1, \tag{108}$$

where the first factor (cf. (80) and (25)) is

$$\mathbf{C} = \begin{pmatrix} & & p & \\ 1 & & v_1 & \\ & \ddots & \vdots & \\ & & \overline{u}_{pp} & \\ & & \vdots & \ddots \\ & & v_m & 1 \end{pmatrix} = \mathbf{E}_p(\mathbf{v}'), \tag{109}$$

and where $\tilde{\mathbf{U}}_i$ $(i > k)$ is defined to be equal to $\mathbf{U}_i$, except for the pth element of its pivot column, which is set to zero.

To show that (108) matches Figure 3, we first notice that the pth column of $\mathbf{C}$ in (109) coincides with the pth column of $\overline{\mathbf{U}}$ in Figure 3; as none of the other factors has a vector different from $\mathbf{e}_p$ in column p, (108) equals Figure 3 in the pth column. Similarly the pth row of $\mathbf{C}$ in (109) coincides with the pth row of $\overline{\mathbf{U}}$ in Figure 3. None of the subsequent factors in (108) will change the row, because we can show that none of the elementary columns in these matrices has a nonzero element in the pth place. For the factors $\tilde{\mathbf{U}}_i$, $(i > k)$, this follows directly from the definition, and in each of the factors $\mathbf{U}_i$, $i < k$, the pth component of its column vector equals the kth component of the column vector of $\mathbf{U}_i^*$, which is zero. Therefore (108) also matches Figure 3 in the pth row. Finally we must show that the general element $\overline{u}_{ij}$, where both i and j differ from p, is the same in Figure 3 and (108). This amounts to show that for $i \neq p$ the two row vectors $\mathbf{e}_i^{\mathrm{T}} \overline{\mathbf{U}}$ and $\mathbf{e}_i^{\mathrm{T}} \mathbf{U}$, where

$\overline{\mathbf{U}}$ is given by (108) and $\mathbf{U}$ by (92), coincide, save for the pth place. We verify this by following $\mathbf{e}_i^{\mathrm{T}}$ through the respective postmultiplications in the expressions (108) and (92). In all nonpivotal positions this row will undergo exactly the same transformations, which again is a consequence of the zero element in the pth row of the factors $\tilde{\mathbf{U}}_i (i > k)$ and $\mathbf{U}_i (i < k)$. This concludes the proof of (108) □.

The inverse form of (108) reads

$$\overline{\mathbf{U}}^{-1} = \mathbf{U}_1^{-1} \ldots \mathbf{U}_{k-1}^{-1} \tilde{\mathbf{U}}_{k+1}^{-1} \ldots \tilde{\mathbf{U}}_m^{-1} \mathbf{C}^{-1}, \tag{110}$$

where the last factor is

$$\mathbf{C}^{-1} = \begin{pmatrix} & & \overset{p}{} & & \\ 1 & & -v_1/\overline{u}_{pp} & & \\ & \ddots & \vdots & & \\ & & 1/\overline{u}_{pp} & & \\ & & \vdots & \ddots & \\ & & -v_m/\overline{u}_{pp} & & 1 \end{pmatrix}. \tag{111}$$

In analogy with $\tilde{\mathbf{U}}_i$ in (108), $\tilde{\mathbf{U}}_i^{-1}$ in (110) can be obtained by zeroing the pth element of the pivotal column of $\mathbf{U}_i^{-1}$. We see that the first $k-1$ factors of (110) are unaffected by the updating procedure, a consequence of the SPUT form of $\mathbf{U}$, cf. (90) and (92). The column vector in $\mathbf{C}^{-1}$ is added as a new vector to the U file. However, in practice we do not remove $\mathbf{U}_k^{-1}$ physically from the file, we only flag it as deleted, such that it virtually equals $\mathbf{I}$ (in the code the flagging is conveniently made by negating the pivot index.)

4.5.3 Interfacing Forrest-Tomlin with simplex

The interfacing of the Forrest-Tomlin $\mathbf{LU}$ method to the simplex procedure outlined in Section 4.1.3 gives rise to amendments to some of the steps, to be listed below.

First we notice that four types of operations are needed to perform the various tasks in the simplex iterations. They are

1. Forward scan of U file, when $\mathbf{U}^{-1}$ is premultiplied by a row vector,
2. Backward scan of U file, when $\mathbf{U}^{-1}$ is postmultiplied by a column vector,
3. Forward scan of L file, when $\mathbf{L}^{-1}$ is postmultiplied by a column vector, and
4. Backward scan of L file, when $\mathbf{L}^{-1}$ is premultiplied by a row vector.

These operations are made by a traditional sparse-matrix technique, holding the U and L files in packed-array form.

Now assume that the current basis-inverse is given by (61). We may then identify the following simplex operations:

BTRAN operation: First, postmultiply the basic cost vector $\mathbf{c}_{\mathrm{B}}^{\mathrm{T}}$ by $\mathbf{U}^{-1}$:

$$\boldsymbol{\gamma}^{\mathrm{T}} = \mathbf{c}_{\mathrm{B}}^{\mathrm{T}} \mathbf{U}^{-1} = \mathbf{c}_{\mathrm{B}}^{\mathrm{T}} \mathbf{U}_1^{-1} \ldots \mathbf{U}_m^{-1} \tag{112}$$

(forward scan of U file). Next, produce a pricing vector $\boldsymbol{\pi}^{\mathrm{T}}$ using $\boldsymbol{\gamma}^{\mathrm{T}}$,

$$\boldsymbol{\pi}^{\mathrm{T}} = \mathbf{c}_{\mathrm{B}}^{\mathrm{T}} \mathbf{B}^{-1} = \boldsymbol{\gamma}^{\mathrm{T}} \mathbf{L}^{-1} = \boldsymbol{\gamma}^{\mathrm{T}} \mathbf{L}_t^{-1} \ldots \mathbf{L}_1^{-1} \tag{113}$$

(backward scan of L file).

PRICE and CHUZC operations: same as in Section 4.1.3.

FTRAN operation: If $\mathbf{a}_{m+q}$ is the column to enter the basis, form the partially updated incoming vector by (69) which in product form gives

$$\mathbf{v} = \mathbf{L}_t^{-1} \ldots \mathbf{L}_1^{-1} \mathbf{a}_{m+q} \tag{114}$$

(forward scan of L file). Complete updating of the incoming vector:

$$\boldsymbol{\alpha}_q = \mathbf{B}^{-1}\mathbf{a}_{m+q} = \mathbf{U}^{-1}\mathbf{v} = \mathbf{U}_1^{-1}\ldots\mathbf{U}_m^{-1}\mathbf{v} \tag{115}$$

(backward scan of U file).

CHUZR operation: same as in Section 4.1.3.

Now that we know which variables will be exchanged we are ready to execute the Forrest-Tomlin updating of the current basis, to transform $\mathbf{B}$ into the adjacent basis matrix $\overline{\mathbf{B}}$ (PIVOT Step in Section 4.1.3). As we saw in Section 4.5.2, this means that we add a new vector to the L-list, cf. (83) – (84), and carry out the transformation leading from $\mathbf{U}^{-1}$ to $\overline{\mathbf{U}}^{-1}$ in (110).

Included in the simplex step must also be an updating of the vector $\boldsymbol{\beta} = \mathbf{B}^{-1}\mathbf{b}$ to its new form $\overline{\boldsymbol{\beta}} = \overline{\mathbf{B}}^{-1}\mathbf{b}$. This is computed from the updated incoming vector $\boldsymbol{\alpha}_q = \{\alpha_{iq}\}$ in (115):

$$\overline{\boldsymbol{\beta}} = \begin{pmatrix} & & & p & \\ 1 & & -\alpha_{1q}/\alpha_{pq} & & \\ & \ddots & \vdots & & \\ & & 1/\alpha_{pq} & & \\ & & \vdots & \ddots & \\ & & -\alpha_{mq}/\alpha_{pq} & & 1 \end{pmatrix} \boldsymbol{\beta}. \tag{116}$$

Equation (116) follows from the relation

$$\overline{\boldsymbol{\beta}} = \overline{\mathbf{B}}^{-1}\mathbf{B}\boldsymbol{\beta} \tag{117}$$

and the equation (35) (or from (19) and (21)).

4.5.4 Implementation

A straightforward implementation of the Forrest-Tomlin procedure could be summarized in the algorithm below:
Initialization part:

1. Start with some triangular factorization of the basis $\mathbf{B}$, with corresponding U and L files and pivot index list.
2. Reset iteration counter to $t = m$, corresponding to the situation immediately after an inversion.
3. Let $\boldsymbol{\beta} = \mathbf{B}^{-1}\mathbf{b}$, where $\mathbf{b}$ is the right-hand side of the system.

Simplex iteration loop:

4. Compute $\boldsymbol{\gamma}^{\mathrm{T}} = \mathbf{c}_{\mathrm{B}}^{\mathrm{T}}\mathbf{U}_1^{-1}\ldots\mathbf{U}_m^{-1}$.
5. Compute $\boldsymbol{\pi}^{\mathrm{T}} = \boldsymbol{\gamma}^{\mathrm{T}}\mathbf{L}_t^{-1}\ldots\mathbf{L}_1^{-1}$.
6. Compute reduced costs $d_j = c_{m+j} - \boldsymbol{\pi}^{\mathrm{T}}\mathbf{a}_{m+j}$.
7. Choose entering column $\mathbf{a}_{m+q}$ with most negative d_j. If no column can be found STOP.
8. Compute $\mathbf{v} = \mathbf{L}_t^{-1}\ldots\mathbf{L}_1^{-1}\mathbf{a}_{m+q}$.
9. Compute $\boldsymbol{\alpha}_q = \mathbf{U}_1^{-1}\ldots\mathbf{U}_m^{-1}\mathbf{v}$.
10. Choose pivot row p by the standard ratio test using $\boldsymbol{\alpha}_q$ and $\boldsymbol{\beta}$.
11. Update $\boldsymbol{\beta}$ by the column vector $\boldsymbol{\alpha}_q$.
12. In the U file, neutralize the elementary column matrix $\mathbf{U}_k$ with pivot index $= p$ (negate the pivot index for that column.)
13. Compute $\sigma\mathbf{r}^{\mathrm{T}} = \mathbf{e}_p^{\mathrm{T}}\mathbf{U}_{k+1}^{-1}\ldots\mathbf{U}_m^{-1}$.

14. In the U file, zero all elements in row p of the post-pivotal elementary matrix factors.
15. Compute $\mathbf{v}' = \mathbf{R}\mathbf{v}$, where $\mathbf{R} = \mathbf{I} - \mathbf{e}_p(\mathbf{e}_p^{\mathrm{T}} - 1/r_p\ \mathbf{r}^{\mathrm{T}})$.
16. Compute new pivot element $\overline{u}_{pp} = v'_p$.
17. Add $\mathbf{C}^{-1} = \mathbf{I} - 1/\overline{u}_{pp}(\mathbf{v}' - \mathbf{e}_p)\mathbf{e}_p^{\mathrm{T}}$ to the U file.
18. Update pivot index list with $p_{t+1} = p$.
19. Add $\mathbf{R}$ to the L file.
20. Increase iteration counter t by one and return to 4 with the updated product forms.
21. End of algorithm.

The steps 4 + 5 form the BTRAN part, 6 is PRICE, 7 is CHUZC, 8 + 9 is FTRAN, 10 is CHUZR, and the remaining steps form together the PIVOT operation.

As pointed out by Forrest and Tomlin [17], a certain amount of computer work may be saved by rearranging the steps described. Their idea is to merge the previous PIVOT updating with the BTRAN for the current simplex step by sandwiching it between operations 4 and 5, and do as much as possible of the U file reading concurrently. The extended BTRAN works as follows:

- Make first part of operation 4 until the pivotal factor $\mathbf{U}_k^{-1}$, which is neutralized (operation 12).
- Continue the U scan till the last factor $\mathbf{C}^{-1}$ with the concurrent execution of operations 13, 14, and operation 4 after $\mathbf{U}_k^{-1}$.
- Operations 15 and 16, and then operation 17 concurrently on completing operation 4 (multiplication by $\mathbf{C}^{-1}$).
- Remaining PIVOT operations 18, 19, and 20.

The growth of the L and U files in the Forrest-Tomlin method will follow a triangular pattern and is generally slower than for the eta file in the product-inverse method.

No actual permutations are necessary: as shown the column permutations cancel, and the row permutations are indirectly taken care of by pointer arrays.

4.6 Miscellaneous features

Until now we have described the fundamental building blocks of LINPROG. In the following we shall add a discussion of more specialized topics. The aim is to give the full background material for understanding how the code works. We shall use the simplex overview in Section 4.1 as the starting point, and step by step introduce the necessary algorithmic modifications. First we shall see how bounds and ranges in the LP formulation are incorporated in a natural way in the simplex framework. After this we describe the simplex initialization procedure in LINPROG, and next how the code finds a first feasible solution. The use of program tolerances as a means to ensure the numerical stability is outlined, and the available variants of matrix scaling are given. Finally, we survey some programming techniques, particularly those related to sparse-matrix representations.

4.6.1 Bounds and ranges

In the standard form of the LP given in Section 4.1.1, the only constraint on the solution vector $\mathbf{x} = \{x_j\}$ was the nonnegativity condition (6). But bounding constraints of the more general type

$$\ell_j \le x_j \le u_j \tag{118}$$

arise quite naturally in many LP applications, and as explained in Chapter 3, LINPROG is able to deal with such constraints. The lower bound constraints $\ell_j \le x_j$ are simply disposed of by a substitution

$$x'_j = x_j - \ell_j, \tag{119}$$

where the bounds for x'_j are

$$\ell'_j = 0 \tag{120}$$

and

$$u'_j = u_j - \ell_j. \tag{121}$$

Assuming that the translation (119) is already carried out, we may drop the primes in x'_j, ℓ'_j, and u'_j. Henceforth we therefore assume $\ell_j = 0$ and consider upper bound constraints only:

$$0 \le x_j \le u_j \quad (j = 1, \ldots, n). \tag{122}$$

Formally we could postulate a restriction (122) for all j, even when x_j is unbounded above, in which case we let $u_j = \infty$. With this convention we can state (122) in vector form:

$$\mathbf{0} \le \mathbf{x} \le \mathbf{u}. \tag{123}$$

In analogy with (8) we order the elements of the bound vector $\mathbf{u}$ according to the current partition in basic and nonbasic variables:

$$\mathbf{u} = \begin{pmatrix} \mathbf{u}_\mathrm{B} \\ \mathbf{u}_\mathrm{N} \end{pmatrix}, \tag{124}$$

such that

$$(\mathbf{u}_\mathrm{B})_i = u_i \quad (i = 1, \ldots, m), \qquad (\mathbf{u}_\mathrm{N})_j = u_{m+j} \quad (j = 1, \ldots, n-m). \tag{125}$$

Of course, constraints like (122) could be realized by adding extra rows to the constraint matrix $\mathbf{A}$ in (5). But it is inefficient to do so, because one of the critical size parameters of an LP is the number of rows m. Instead we extend the simplex idea by allowing a nonbasic variable x_j to be equal to *either* of its bounds $x_j = 0$ or (if $u_j < \infty$) $x_j = u_j$.

In the bounded simplex procedure the formulas (7) – (15) are still valid. The restriction (16) should be modified to

$$\mathbf{0} \le \mathbf{x}_\mathrm{B} \le \mathbf{u}_\mathrm{B}, \quad \mathbf{0} \le \mathbf{x}_\mathrm{N} \le \mathbf{u}_\mathrm{N}. \tag{126}$$

Also the expression (17) for the current basic solution must be altered. Let us divide the total nonbasic index set $\{1, \ldots, n-m\}$ in the lower bound set

$$L = \{j \mid x_{m+j} \text{ nonbasic at zero}\} \tag{127}$$

and the upper bound set

$$U = \{j \mid x_{m+j} \text{ nonbasic at } u_{m+j}\}. \tag{128}$$

Then (17) should be replaced by

$$\mathbf{x}_\mathrm{B} = \tilde{\boldsymbol{\beta}}, \tag{129}$$

where we have introduced the "effective" $\boldsymbol{\beta}$-vector

$$\tilde{\boldsymbol{\beta}} = \{\tilde{\beta}_i\} = \boldsymbol{\beta} - \sum_{j \in U} u_{m+j} \boldsymbol{\alpha}_j, \tag{130}$$

with β defined in (12) and α_j in (20). Using L and U, the objective function z in (14) can be written

$$z = \mathbf{c}_{\mathrm{B}}^{\mathrm{T}}\beta + \sum_{j\in L} d_j x_{m+j} + \sum_{j\in U} d_j x_{m+j}, \tag{131}$$

where $\mathbf{d} = \{d_j\}$ is the reduced-cost vector defined by (18). Suppose now that we have a basic feasible solution for which the condition

$$\forall j \in L \quad d_j \geq 0 \quad \text{and} \quad \forall j \in U \quad d_j \leq 0 \tag{132}$$

holds. Then (131) shows that z is at its minimum; it will not decrease, if we increase x_{m+j} from zero ($j \in L$), or if we decrease x_{m+j} from u_{m+j} ($j \in U$). Thus (132) is a sufficient condition for optimality. If (132) does not hold, then either

$$\exists j \in L \quad d_j < 0, \tag{133}$$

or

$$\exists j \in U \quad d_j > 0. \tag{134}$$

In both cases we may improve on z by introducing a nonbasic variable x_{m+j} into the basis; in the former case it should be increased from zero, and in the latter case decreased from u_{m+j}.

Also the selection of the leaving variable becomes somewhat more complicated than in the algorithm with no upper bounds. This is because a basic variable $(\mathbf{x}_{\mathrm{B}})_i = x_i$ may reach either its lower bound 0 or its upper bound $(\mathbf{u}_{\mathrm{B}})_i = u_i$. Moreover, we must consider the possibility of the entering variable reaching its opposite bound and thus becoming nonbasic anew. Which of these events will first take place depends on the respective step lengths θ_1, θ_2, and θ_3, of the entering variable x_{m+q}. We see from (15) and (130) that the critical steps should be determined from the expression

$$\mathbf{x}_{\mathrm{B}} = \tilde{\beta} - x_{m+q}\alpha_q, \tag{135}$$

if $q \in L$, and from

$$\mathbf{x}_{\mathrm{B}} = \tilde{\beta} + (u_{m+q} - x_{m+q})\alpha_q, \tag{136}$$

if $q \in U$. We then compute θ_1 as follows:

$$q \in L: \quad \theta_1 = \min\left\{ \tilde{\beta}_i/\alpha_{iq} : i = 1,\ldots,m \text{ and } \alpha_{iq} > 0 \right\}, \tag{137}$$

$$q \in U: \quad \theta_1 = \min\left\{ -\tilde{\beta}_i/\alpha_{iq} : i = 1,\ldots,m \text{ and } \alpha_{iq} < 0 \right\}; \tag{138}$$

the step θ_2 is similarly computed as:

$$q \in L: \quad \theta_2 = \min\left\{ (\tilde{\beta}_i - u_i)/\alpha_{iq} : i = 1,\ldots,m \text{ and } \alpha_{iq} < 0 \right\}, \tag{139}$$

$$q \in U: \quad \theta_2 = \min\left\{ (u_i - \tilde{\beta}_i)/\alpha_{iq} : i = 1,\ldots,m \text{ and } \alpha_{iq} > 0 \right\}. \tag{140}$$

In (137) – (140) we use the convention

$$\min\{\emptyset\} = \infty. \tag{141}$$

We see that (137) corresponds to (21) in the standard algorithm. Because we have a basic feasible solution,

$$0 \leq x_i \leq u_i, \tag{142}$$

the two steps θ_1 and θ_2 cannot be negative. Defining

$$\theta_3 = u_{m+q}, \tag{143}$$

it is clear that the x_{m+q}-step for the first event to happen will be given by

$$\theta = \min\{\theta_1, \theta_2, \theta_3\}. \tag{144}$$

As in the standard algorithm, $\theta = \infty$ means an unbounded solution. In the two cases $\theta = \theta_1$ and $\theta = \theta_2$ we carry out a pivot step, which comprises a basis exchange operation (Forrest-Tomlin). In addition we must compute the new basic solution $\overline{\mathbf{x}}_B$. For the standard algorithm we have $\overline{\mathbf{x}}_B = \overline{\boldsymbol{\beta}}$, where $\overline{\boldsymbol{\beta}}$ was given by (116). From (135) – (140) it can be shown that with the upper bound algorithm we should use the following more general form for $\overline{\mathbf{x}}_B$:

$$\overline{\mathbf{x}}_B = \overline{\boldsymbol{\beta}}_0 + \begin{pmatrix} & & p & \\ 1 & & -\alpha_{1q}/\alpha_{pq} & \\ & \ddots & \vdots & \\ & & 1/\alpha_{pq} & \\ & & \vdots & \ddots \\ & & -\alpha_{mq}/\alpha_{pq} & 1 \end{pmatrix} (\tilde{\boldsymbol{\beta}} - \tilde{\boldsymbol{\beta}}_0), \tag{145}$$

where

$$\tilde{\boldsymbol{\beta}}_0 = \begin{cases} \mathbf{0} & \text{if } x_p \text{ hits zero} \\ u_p \mathbf{e}_p & \text{if } x_p \text{ hits } u_p \end{cases} \tag{146}$$

and

$$\overline{\boldsymbol{\beta}}_0 = \begin{cases} \mathbf{0} & \text{if } q \in L \\ u_{m+q} \mathbf{e}_p & \text{if } q \in U \end{cases} \tag{147}$$

In the third case, $\theta = \theta_3$, the nonbasic variable x_{m+q} goes to its opposite bound. This must be recorded, as must also the induced change in the current basic solution; the new solution becomes (cf. (135) – (136)):

$$\overline{\mathbf{x}}_B = \tilde{\boldsymbol{\beta}} - u_{m+q} \boldsymbol{\alpha}_q \quad \text{for } q \in L, \tag{148}$$

$$\overline{\mathbf{x}}_B = \tilde{\boldsymbol{\beta}} + u_{m+q} \boldsymbol{\alpha}_q \quad \text{for } q \in U. \tag{149}$$

We can now summarize the upper bound simplex procedure in algorithmic form. Compared to the standard algorithm in Section 4.1.3, the only steps to modify are CHUZC, CHUZR, and the $\boldsymbol{\beta}$-updating part of PIVOT:

- CHUZC Step — Choose the entering nonbasic variable ($j = q$) as one from L with negative reduced cost d_j, or one from U with positive reduced cost d_j. (In LINPROG we choose one with maximum $|d_j|$.) If no candidate q can be found, the current solution is optimal.
- CHUZR Step — Use one of the ratios (137) or (138) to compute a step θ_1 for x_{m+q} taking a basic variable x_i to zero. Also we use one of the ratio rules (139) or (140) to compute a step θ_2 taking x_i to its upper bound u_i. Put $\theta_3 = u_{m+q}$; the critical step θ is then the minimum of θ_1, θ_2, and θ_3. If $\theta = \theta_3$, we record the bound shift of the nonbasic variable and the shift of the basic solution as expressed by (148) or (149), and then we return to the CHUZC Step. If $\theta = \infty$, we have an unbounded solution. Otherwise we proceed to the PIVOT Step.
- PIVOT Step — Make the basis matrix exchange. Update the current basic solution by using (145). Return to the BTRAN Step.

Recall from Chapter 3, that LINPROG is able to handle a BOUNDS section with bound types UP, LO, FX, FR, PL, and MI. We have so far concentrated on the UP-facility, and this is in fact the only one which took some difficulty to implement. As already mentioned, LO is taken care of by substitution. A variable of type FX is permanently locked out from the basis, and a FR variable is locked into the basis. PL is obtained by default, and MI is realized by negating the variable.

When implementing RANGES constraints of the type given in (3) in Chapter 3, LINPROG takes advantage of the upper bound framework outlined previously. It transforms (3) to an equality restriction

$$\sum a_j x_j - y = 0 \tag{150}$$

by introducing a new variable y, which might be thought of as a "slack" variable associated with range constraints. The variable y is bounded below and above,

$$\ell \leq y \leq u, \tag{151}$$

and the way such a constraint is handled in LINPROG has already been discussed at length.

4.6.2 Simplex initialization: CRASH and zero-slack elimination

Before the standard simplex algorithm can work, we must have a basic feasible solution. LINPROG achieves this goal in several stages. Here we shall describe the first two of these: the "CRASH" procedure for setting up an initial basis, and the "Phase 0" procedure for removal of zero slacks from the basis. In Section 4.6.3 we will describe how the infeasibilities are driven to zero by a special pass of simplex called "Phase 1".

In the description of the ROWS Section output in Section 3.4 we saw that not only will each column in the constraint matrix be associated with a variable, but so also will each row. LINPROG uses the same principle in its internal organization of the computations. To discern the row variables from the physical variables (the latter being the "structural" variables), the former are often called "logical variables" or "slack variables".

In the LP formulation (1) in Chapter 2 we may assume that the restrictions $\mathcal{R}_i$ are either $\leq$ or $=$, because, as mentioned in Section 4.1.1, the $\geq$-restrictions are converted to $\leq$-restrictions, and the free rows are deleted from the constraint matrix. To each $\leq$-restriction there corresponds a nonnegative slack variable, and to each equality restriction a "zero-slack" variable.

Adopting this convention, LINPROG augments the original constraint matrix with a unit matrix $\mathbf{I}$, which comes from the slack variables. It is this augmented matrix that corresponds to the $m \times n$ constraint matrix $\mathbf{A}$ in the formulation (4) – (6). Rather than (6) we use the constraint (123) for $\mathbf{x}$. Notice that zero slacks may be considered as bounded variables with zero as the common lower and upper bound. Basic zero slacks will normally be infeasible.

Our first task will be to select a subset of m independent columns from $\mathbf{A}$ to form the initial basis matrix $\mathbf{B} = \mathbf{B}_0$. Assuming that all nonbasic variables start at zero, equations (17) and (12) give us the first basic solution. It might be infeasible, but we are willing to accept this in the initial stage.

Of course, an easy choice would be to select the all-slack initial basis matrix $\mathbf{B}_0 = \mathbf{I}$. The inversion in (12) would be trivial, and we would get the initial solution $\mathbf{x}_B = \mathbf{b}$. But this solution might very well contain many infeasible elements, which afterwards would entail much work in Phase 1. Fortunately we can do better than this. Following common practice we choose a starting basis matrix $\mathbf{B}_0$ that is as close to feasibility as possible and is also (permuted) triangular. This is the CRASH procedure. A triangular basis matrix is still easy to invert. After experimenting with some CRASH variants we found that for a broad class of test problems a good strategy is to choose as many feasible slacks and structurals as possible. We do this by a recursive scanning of $\mathbf{A}$ for triangularity, starting with the unit vectors from the feasible slacks. When no more feasible slacks or structurals are available for a triangular $\mathbf{B}$, we are forced to fill the remaining places with infeasible slacks.

After setting up various indicators for bound status and for the nonbasic variables, LINPROG performs the initial inversion. After this the code is ready to enter the next stage, which is the elimination of zero slacks from the basis (Phase 0). One can hope that the CRASH procedure was successful in keeping most zero slacks out of basis, but in general there will be some of them in the basis. Phase 0 resembles the other simplex phases in many respects. The basis exchange mechanism is the Forrest-Tomlin update scheme described in Section 4.5. The difference lies in the selection criteria for departing and entering variables. In Phase 0 we first choose the departing variable. The criteria for selecting the entering variable are first that it must not be an already eliminated zero slack (or any other fixed variable), and next that the prospective pivot element ($\overline{u}_{pp}$ in (81)) be nonzero (otherwise the new column would be linearly dependent on those already in the basis). In practice we scan each available column until the condition

$$|\overline{u}_{pp}| \geq \epsilon_0 \tag{152}$$

is fulfilled, in which case we accept the column; if (152) is not satisfied for any column, LINPROG uses the column with greatest $|\overline{u}_{pp}|$ (ϵ_0 corresponds to the parameter ZERPIV mentioned in Section 4.6.5).

Suppose now that a successful pivot has been found. We then mark the eliminated zero slack as unavailable for future re-entrance into the basis and proceed to the Forrest-Tomlin exchange procedure. On the other hand, assume that no pivot element $\overline{u}_{pp} \neq 0$ could be found. Depending on the corresponding component of the transformed right-hand side β, there are two possibilities: If the component is $\neq 0$, we have detected an infeasibility. If it is zero, the corresponding restriction is redundant. In that case it is impossible to remove the zero slack from the basis, and it must stay locked into the basis throughout the simplex process.

Also in Phase 0 we invoke re-inversions at regular intervals of the iteration counter.

4.6.3 Finding a first feasible solution: Phase 1

Let us recapitulate the main passes of simplex used in LINPROG:

- CRASH: Provide an initial triangular basis.
- Phase 0: Try to remove zero slacks from the basis.
- Phase 1: Establish feasibility (or demonstrate infeasibility).
- Phase 2: Establish optimality.

After CRASH and Phase 0, LINPROG enters Phase 1, in which the infeasibilities are sought removed. There are several variants of Phase 1 in use in different simplex codes. For example, the zero slack eliminations, which we segregated out as a special Phase 0, could instead be integrated in Phase 1. We included Phase 0 to improve the efficiency on LP problems with many equality restrictions; notice in this context that LINPROG can be used to solve n linear equations in n unknowns.

A central feature in Phase 1 is the use of a special objective function, called the "sum of infeasibilities", or SINF, instead of the objective function $\mathbf{c}^{\mathrm{T}}\mathbf{x}$ in (4). We minimize SINF by the simplex technique; if the minimum is zero, feasibility is achieved. Otherwise the LP is infeasible. In this way Phase 1 is just a variant of the simplex procedure discussed in Sections 4.1 and 4.6.1. The main difference is that in Phase 1 we allow basic solutions to be infeasible. In other implementations the basic feasibility is maintained formally by introducing so-called "artificial variables". Such variables are not used in LINPROG.

When explaining our Phase 1 technique, which is inspired by the code of Bartels *et al.* [18], and which also resembles the method described by Nazareth [4], our

starting point will be the bounded simplex procedure given in Section 4.6.1. The first step is to give a precise definition of SINF. This definition is dynamical in the sense that it reflects the infeasibility status of the current basic solution $\mathbf{x}_B = \boldsymbol{\beta}$ given by (17) and (12). (Admittedly, the upper-bounding algorithm may induce a correction to $\boldsymbol{\beta}$ (cf. (130)), but for notational convenience we assume that this correction is already included in $\boldsymbol{\beta}$.) Let us partition the basic index set

$$B = \{1, \ldots, m\} \tag{153}$$

in index sets for "feasibility", "infeasibility below lower bound", and "infeasibility above upper bound":

$$B = F \cup I^- \cup I^+. \tag{154}$$

To define the sets in (154) formally, assume that the ith basic variable $x_i = \beta_i$ is bounded below by ℓ_i and above by u_i (Though LINPROG works with the lower bound $\ell_i = 0$ internally, we keep ℓ_i here as a general parameter to make the exposition clearer.) We then have

$$F = \{i \in B | \, \ell_i \leq \beta_i \leq u_i\}, \tag{155}$$

$$I^- = \{i \in B | \, \beta_i < \ell_i\}, \tag{156}$$

and

$$I^+ = \{i \in B | \, \beta_i > u_i\}. \tag{157}$$

We can depict the possible feasibility states in a diagram as shown in Figure 5. The three cross marks represent the typical states for a basic variable. As a special

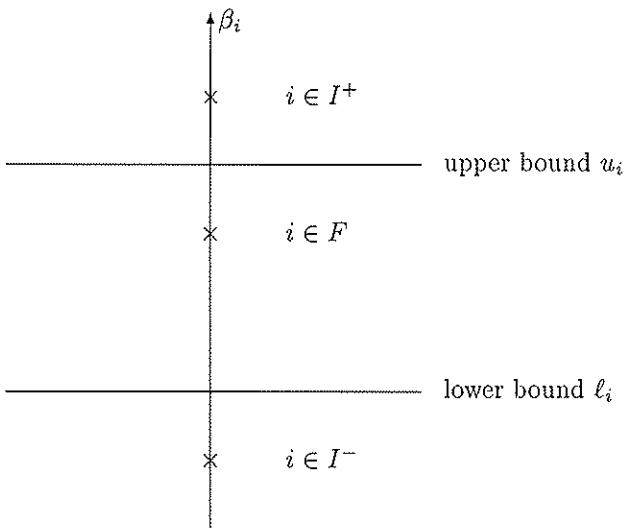


Figure 5. Possible feasibility states for basic variable $x_i = \beta_i$

case we may have $u_i = +\infty$. With the notation introduced we define

$$\text{SINF} = \sum_{i \in I^-} (\ell_i - x_i) + \sum_{i \in I^+} (x_i - u_i), \tag{158}$$

justifying the phrase "sum of infeasibilities" for SINF. We may also write SINF as a linear function similar to (4). To do this, we introduce the "infeasibility

cost vector" $\mathbf{c}_\mathrm{I}$. This n-vector is partitioned in a basic m-vector and a nonbasic $(n-m)$-vector as in (8):

$$\mathbf{c}_\mathrm{I} = \begin{pmatrix} \mathbf{c}_\mathrm{BI} \\ \mathbf{c}_\mathrm{NI} \end{pmatrix}. \tag{159}$$

As nonbasic variables are always feasible we let

$$\mathbf{c}_\mathrm{NI} = \mathbf{0}. \tag{160}$$

The ith component of $\mathbf{c}_\mathrm{BI}$ is set to 0, -1, or $+1$, according to the following rules:

$$i \in F \quad \Rightarrow \quad (\mathbf{c}_\mathrm{BI})_i = \ \ 0, \tag{161}$$

$$i \in I^- \quad \Rightarrow \quad (\mathbf{c}_\mathrm{BI})_i = -1, \tag{162}$$

$$i \in I^+ \quad \Rightarrow \quad (\mathbf{c}_\mathrm{BI})_i = +1. \tag{163}$$

We see that apart from a constant, SINF is the inner product of $\mathbf{c}_\mathrm{I}$ with $\mathbf{x}$:

$$\mathrm{SINF} = \mathbf{c}_\mathrm{I}^\mathrm{T}\mathbf{x} \ + \sum_{i \in I^-} \ell_i - \sum_{i \in I^+} u_i. \tag{164}$$

In contrast to (4), the linear form (164) is not fixed, but changes from iteration to iteration. Nevertheless, the main simplex principle of reducing the objective function from one iteration to the next still holds, such that we expect Phase 1 to be finished in a finite number of steps, ignoring the possibility of cycling.

We may also speak of the *number* of infeasibilities

$$\mathrm{NINF} = |I^- \cup I^+|. \tag{165}$$

When feasibility is attained, then I^- and I^+ are empty; hence NINF $= 0$, while (158) gives SINF $= 0$. Also $B = F$, and by (160) and (161) $\mathbf{c}_\mathrm{I} = \mathbf{0}$. We therefore deduce that

$$\mathbf{c}_\mathrm{I} = \mathbf{0} \quad \Longleftrightarrow \quad \mathrm{NINF} = 0 \quad \Longleftrightarrow \quad \mathrm{SINF} = 0, \tag{166}$$

such that the fulfillment of any of these three conditions means feasibility, in which case Phase 1 terminates. Otherwise we proceed by computing the "infeasibility pricing vector"

$$\boldsymbol{\pi}_\mathrm{I}^\mathrm{T} = \mathbf{c}_\mathrm{BI}^\mathrm{T}\mathbf{B}^{-1} \tag{167}$$

by the same BTRAN operation as used for the normal pricing vector (13); after this the vector of "reduced infeasibility costs"

$$\mathbf{d}_\mathrm{I}^\mathrm{T} = \mathbf{c}_\mathrm{NI}^\mathrm{T} - \boldsymbol{\pi}_\mathrm{I}^\mathrm{T}\mathbf{N} = -\boldsymbol{\pi}_\mathrm{I}^\mathrm{T}\mathbf{N} \tag{168}$$

is computed in analogy with (18). We can now argue in precisely the same way as in Sections 4.1.2 and 4.6.1 to obtain conditions for SINF to be minimal, and, if it is not minimal, to find an entering variable to reduce it. We find that if $(\mathbf{d}_\mathrm{I})_j \geq 0$ for all nonbasic variables at lower bound and $(\mathbf{d}_\mathrm{I})_j \leq 0$ for all nonbasic variables at upper bound, then SINF is minimal. Formally this condition can be stated as in (132):

$$\forall j \in L \ \ (\mathbf{d}_\mathrm{I})_j \geq 0 \quad \text{and} \quad \forall j \in U \ \ (\mathbf{d}_\mathrm{I})_j \leq 0. \tag{169}$$

We know that SINF > 0, and if (169) holds, our LP must be infeasible. If not we use exactly the same CHUZC procedure as in Section 4.6.1 to select a variable x_{m+q} to be introduced into the basis.

The more difficult part of Phase 1 is to devise a strategy for selecting a variable to leave the basis (CHUZR). Again a number of variants are in use in different simplex codes. The method in LINPROG forms a natural generalization of the bounded simplex algorithm for feasible solutions given in Section 4.6.1. It works well for the test problems we have studied.

We assume that the CHUZC procedure by now has selected the variable x_{m+q} to be introduced into the basis. We must then choose an index $i = p$ from the set

B in (153) identifying a variable to leave the basis. Consider the updated incoming vector (cf. (115) and (34)):

$$\alpha_q = \{\alpha_{iq}\}_{i=1,\ldots,m} = \mathbf{B}^{-1}\mathbf{a}_{m+q}. \tag{170}$$

First we notice that a leaving candidate $i = p$ must have its pivot element $\alpha_{pq} \neq 0$. We therefore introduce the index set B_1 of candidates available for pivoting:

$$B_1 = \{i \in B | \, \alpha_{iq} \neq 0\}, \tag{171}$$

and in the following discussions we rule out all candidates $i \notin B_1$. We can partition B_1 in the same way as we did for B in (154):

$$B_1 = F_1 \cup I_1^- \cup I_1^+, \tag{172}$$

where

$$F_1 = F \cap B_1 = \{i \in B_1 | \, \ell_i \leq \beta_i \leq u_i\}, \tag{173}$$

$$I_1^- = I^- \cap B_1 = \{i \in B_1 | \, \beta_i < \ell_i\}, \tag{174}$$

and

$$I_1^+ = I^+ \cap B_1 = \{i \in B_1 | \, \beta_i > u_i\}. \tag{175}$$

Thus F_1 is the feasible part of B_1. The infeasible part is

$$I_1 = I_1^- \cup I_1^+. \tag{176}$$

Making x_{m+q} basic means that either it rises from its lower bound ℓ, or it decreases from its upper bound u. If θ is the absolute value of the x_{m+q}-step in both cases, we can calculate the change in the ith basic solution component using (19), which holds for small enough $\theta \geq 0$:

$$x_i = \beta_i \mp \theta\alpha_{iq}, \tag{177}$$

where the upper sign applies if x_{m+q} rises, and the lower if it decreases. As we have already excluded the case $\alpha_{iq} = 0$, we know that the basic variable x_i will move from one of the typical cross positions for β_i in Figure 5, when θ increases from 0. Whether (177) represents an increase or a decrease of x_i from β_i depends on the bound of x_{m+q} and the sign of α_{iq}. For each candidate $i \in B_1$ let us define a *direction indicator* D_i to be $\uparrow$ ("up") or $\downarrow$ ("down"), by the following rule:

$$D_i = \begin{cases} \downarrow & \text{if} \quad (\alpha_{iq} > 0 \wedge x_{m+q} = \ell) \vee (\alpha_{iq} < 0 \wedge x_{m+q} = u) \\ \uparrow & \text{if} \quad (\alpha_{iq} < 0 \wedge x_{m+q} = \ell) \vee (\alpha_{iq} > 0 \wedge x_{m+q} = u). \end{cases} \tag{178}$$

Using this direction indicator, we partition the infeasible set I_1 in (176) in another way:

$$I_1 = I_{\text{better}} \cup I_{\text{worse}}, \tag{179}$$

where

$$I_{\text{better}} = \{i \in I_1^+ | D_i = \downarrow\} \cup \{i \in I_1^- | D_i = \uparrow\}, \tag{180}$$

and

$$I_{\text{worse}} = \{i \in I_1^+ | D_i = \uparrow\} \cup \{i \in I_1^- | D_i = \downarrow\}. \tag{181}$$

The set I_{better} represents infeasible candidates moving towards feasibility, while I_{worse} represents infeasible candidates moving further away from feasibility. It is clear that we should never accept a candidate from I_{worse}:

$$p \notin I_{\text{worse}}. \tag{182}$$

We therefore concentrate on the candidate set

$$C = F_1 \cup I_{\text{better}}, \tag{183}$$

and for each $i \in C$ we compute certain critical step lengths for x_{m+q}, taking the corresponding basic variable x_i to either of its bounds ℓ_i or u_i:

$$i \in F_1: \qquad \theta_i = \begin{cases} (u_i - \beta_i)/|\alpha_{iq}| & \text{for } D_i = \uparrow \\ (\beta_i - \ell_i)/|\alpha_{iq}| & \text{for } D_i = \downarrow, \end{cases} \tag{184}$$

$$i \in I_{\text{better}}: \qquad \theta_{\text{in}} = \begin{cases} (\beta_i - u_i)/|\alpha_{iq}| & \text{for } D_i = \downarrow \\ (\ell_i - \beta_i)/|\alpha_{iq}| & \text{for } D_i = \uparrow, \end{cases} \tag{185}$$

and

$$i \in I_{\text{better}}: \qquad \theta_{\text{if}} = \begin{cases} (\beta_i - \ell_i)/|\alpha_{iq}| & \text{for } D_i = \downarrow \\ (u_i - \beta_i)/|\alpha_{iq}| & \text{for } D_i = \uparrow. \end{cases} \tag{186}$$

Next we calculate

$$\theta_1' = \min\{\theta_i : i \in F_1\}, \tag{187}$$

$$\theta_1'' = \min\{\theta_{\text{if}} : i \in I_{\text{better}}\}, \tag{188}$$

and from this

$$\theta_1 = \min\{\theta_1', \theta_1''\}. \tag{189}$$

We also compute

$$\theta_2 = \max\{\theta_{\text{in}} : i \in I_{\text{better}}\}. \tag{190}$$

In these formulas, θ_i takes a feasible x_i to its bound, θ_{in} takes an infeasible x_i to its nearest bound, whereas θ_{if} takes an infeasible x_i to its farthest bound. It may happen that $F_1 = \emptyset$, in which case we use the convention (141). On the other hand I_{better} is non-empty because SINF > 0 and $d_q \neq 0$. Moreover we let

$$\theta_3 = u - \ell, \tag{191}$$

i.e. the difference between the upper and lower bounds for the entering variable. Then finally, we use the step

$$\theta = \min\{\theta_1, \theta_2, \theta_3\}. \tag{192}$$

If θ_3 becomes blocking, the leaving and entering variables coincide.

To state it in another way, we can characterize the properties of our strategy as follows: The leaving variable goes to one of its bounds. No feasible variable goes infeasible, hence the number of infeasibilities NINF is monotonically decreasing. No infeasible variable can cross the feasible range and again go infeasible. Within these constraints we eliminate as many infeasibilities as we possibly can. This Phase 1 strategy does not guarantee that the sum of infeasibilities SINF be monotonic decreasing, too. It would be so if we always were to stop at the nearest boundary; this is a consequence of the way we select the entering variable from the "reduced infeasibility costs" in CHUZC. But when we allow an infeasible variable x_i to go to its farthest bound, there exists a possibility that SINF might increase: the initial decrease in SINF could be outweighed by a subsequent increase, because when x_i moves into the feasible range between the bounds, we suddenly lose its contribution to the decrease in SINF. Notice however, that when this happens, NINF certainly decreases, which is important for the convergence of the procedure.

Let us summarize the Phase 1 CHUZR in compressed algorithmic form, as we did it for the Phase 2 CHUZR in Section 4.6.1:

- CHUZR Step in Phase 1 — Compute the least step θ_1' taking a feasible basic variable to its bound by (187). Compute the least step θ_1'' taking an infeasible variable to its farthest bound by (188). Find the minimum θ_1 of these two

steps. Compute the largest step θ_2 taking an infeasible basic variable to its nearest bound by (190). Compute the difference θ_3 between the upper and lower bounds for the entering variable. Take the least of the three steps θ_1, θ_2, and θ_3, to be the resulting step θ of the entering variable. If $\theta = \theta_3$ make the entering variable nonbasic at its opposite end. Otherwise identify the leaving variable and proceed to the PIVOT Step.

We have not paid attention to problems arising from degeneracies, which in connection with rounding errors may introduce serious troubles in practical computations. A common means of trying to improve the robustness of the CHUZR procedure is to use a two-pass version of the procedure, where the bounds are slightly perturbed in the first pass, but treated exactly in the second. This technique, which dates back to Harris [19], has also been tried in LINPROG, but we did not find the results sufficiently rewarding to justify the coding complications. Instead, we use feasibility tolerances in a much simpler way (Section 4.6.5).

Degeneracies of the type $u_i = \ell_i$ for basic variables will not occur in Phase 1. Like zero slacks such variables are considered as fixed variables by LINPROG, and no fixed variable can be basic and available for pivoting in Phase 1.

We have also tried another Phase 1 strategy, the so-called FEWPHI method described by Maros [20] and by Nazareth [4]. The idea is to relax the condition that feasible variables always stay feasible after a transformation, as long as SINF does not increase. In fact we can make a local optimization of SINF whose graph is a piecewise linear convex curve. With this strategy SINF would always be monotone though NINF need not be so. Our computer experiments show that FEWPHI is capable to reduce the Phase 1 iteration count for some test problems, but not for others. More tests are necessary before we would embark on implementing it in LINPROG.

When Phase 1 terminates with a feasible solution, we drop the infeasibility cost function and return to the genuine objective function of (4). This will be minimized in "Phase 2" of simplex, where we use the mechanisms described in Section 4.1 together with the upper-bounding amendments in Section 4.6.1. In fact the above description of the Phase 1 CHUZR algorithm also holds for Phase 2, where now all the "infeasible" sets are empty.

4.6.4 Pricing strategies. Cycling.

As described in Sections 4.1.2 and 4.6.1, LINPROG uses a simple criterion for selecting the entering variable: First the reduced costs d_j are calculated by (18) (operation PRICE), and next the code seeks out a column with maximum $|d_j|$ among the candidates with the proper sign of d_j (operation CHUZC). Taken together, PRICE and CHUZC form our pricing strategy. Many refinements of this strategy are in use in commercial LP software. Some of them will be mentioned here with comments on their potential usefulness for enhancing the performance of LINPROG.

The advanced pricing methods can be divided in two classes. In the first class of methods an attempt is made to reduce the number of simplex iterations by modifying the CHUZC selection criterion based on d_j. This requires an increase of the work per iteration, however. The second class of methods seeks to reduce the average work per iteration at the expense of some increase of the iteration count.

To the first class belong the *steepest-edge* pricing methods, of which the DEVEX scheme by Harris [19] and the method of Goldfarb and Reid [21] are the best known. The simple LINPROG pricing based on the d_j-criterion corresponds to finding an edge of the polytope (Section 4.1.2) with maximum absolute value of

the gradient of the objective function, measured in the *reference space* of the current nonbasic variables. This reference space changes at each iteration. Steepest-edge pricing, on the other hand, uses gradients in a *fixed* reference space. Harris uses a reference space spanned by the structural variables of the LP. She computes the gradients in this space by attaching proper weight parameters w_j to the reduced costs d_j, such that d_j/w_j rather than d_j are used for selecting the incoming variable. The weights w_j, being costly to compute afresh, are estimated from previous iterates using a heuristic recursion formula. Goldfarb and Reid use a similar approach. Their method differs from that of Harris by the use of a reference space spanned by all the LP variables, and the use of an exact recursion formula for the weights. Both methods affect a dynamic scaling of the columns. We have experimented with the Goldfarb and Reid method in LINPROG for selected sample problems. As expected, the result was a significant reduction of the iteration count (typically about 40 per cent), but this gain was offset by the extra work involved in the pricing operation, so that no overall improvement could be measured. Moreover, the recurrence formula seemed to induce numerical instability. We have not tried Harris's scheme, but this is generally rated equal to that of Goldfarb and Reid in efficiency. Of course, such comparisons depend a lot on the computer environment. Our conclusion was that it is not worthwhile to use steepest-edge pricing in LINPROG.

Another way to reduce the number of simplex iterations is by "multiple-target" pricing, where more than one objective function is used concurrently. For example, in Phase 1 we could compute reduced costs for *both* the sum of infeasibilities *and* the "genuine" objective function and find an incoming variable which decreases both of them. (The so-called "Big-M" method [8, 4] forms a variant of this principle; its use, however, is discouraged for numerical reasons.) It is also possible to select a secondary objective function in such a way that degeneracies are impeded. This technique, which is described by Nazareth [4], has been tried in LINPROG, but without sufficiently encouraging results. Otherwise, multiple-target pricing methods have not been tried by us.

Let us briefly consider the other class of pricing methods, in which one seeks to lessen the average work in the pricing process. The motivation for such methods is that the full product $\boldsymbol{\pi}^{\mathrm{T}}\mathbf{N}$ in (18) represents $n - m$ inner-product operations and may therefore be costly to compute for big problems with many nonbasic variables.

With *partial pricing* only a portion of the nonbasic columns in $\mathbf{N}$ is scanned at each iteration to find an incoming candidate. At the next iteration another portion is scanned, for example by starting where the last scan ended. Experiments with partial pricing in LINPROG did not show significant improvement of performance, but the reason could be that few of our large test cases have $n \gg m$. Indeed, for such problems Bixby *et al.* [22] reports favourable results with the CPLEX code by using a combination of steepest-edge and partial pricing.

With *multiple pricing* one seeks out a number of promising candidates, say five, with "good" reduced costs, and processes them together in extended FTRAN, CHUZR, and PIVOT operations. Only one candidate, of course, is selected to be introduced into the basis, but for the next iteration only the remaining four columns are considered. The process continues as long as the reduced costs admit it. In this way a subproblem is defined, within which *minor iterations* take place. In the next *major iteration* we define a new subproblem with five new candidates, and so on. This technique should reduce the total simplex work. It is also possible to make a local CHUZR optimization within the subproblem by finding a step θ which reduces the objective function as much as possible. Preliminary experiments with multiple pricing indicate that this technique might indeed significantly improve the efficiency in LINPROG. Maybe the best strategy would be to combine multiple

pricing with partial pricing. The present version of LINPROG contains no multiple pricing.

We shall also comment on *cycling*. It was mentioned in the description of the standard simplex method in Section 4.1.2 that basic solutions may become degenerate such that one or more basic variables are zero (or more generally, equal to one of their bounds). Then there exists a possibility, that the step size θ in (21) equals zero, with the consequence that the objective function is unaltered. After a series of such exchanges between degenerate solutions we might again return to the same set of basic variables, and we have an instance of cycling.

Though it is generally accepted that cycling is an unlikely phenomenon, we have seen some instances in LINPROG and its predecessors. The use of the various simplex modifications we have described, viz. the use of a dynamical sum-of-infeasibility form SINF and of the upper-bounding algorithm, increase the risk of cycling. Systematic methods for cycle protection are given in the literature. For example, Garfinkel and Nemhauser [23] describe a technique based on a lexicographic ranking rule for vectors; Gill *et al.* [24] discuss practical experiences from using an "anti-cycling procedure" in LP computations.

Apart from genuine cycling there is also the risk of "stalling" with very little progress or no progress at all over say hundreds or even thousands of iterations. For this reason a no-progress check is built into the program. As stated in Section 3.2 this is controlled by the value of NOPROG.

In LINPROG we preferred to use a few specific devices to counteract the onset of cycling or stalling. These are attached to the CHUZR procedure and will be mentioned in the subsequent discussion of LP tolerances. Still we cannot reject the possibility that cycling may strike unexpectedly in some problem or another.

4.6.5 Scaling, LP tolerances, and numerical stability.

When an LP is solved by a computer, the finite precision arithmetic will inevitably induce rounding errors. Such errors are well-known from solution of equations, but their impact is much more unpredictable when inequalities are considered. Following common practice, LINPROG uses a set of *tolerances* to provide margins of error when comparisons are made between two numbers or between a number and zero, as required in the simplex procedure.

To make the use of such tolerances meaningful, the LP problem should first be brought on a suitable standard form. Therefore, before starting the simplex algorithm, LINPROG normally makes a *scaling* of the left-hand side of the LP system (1). We have seen in Section 3.2, that there are several possible scaling options, governed by the numerical keyword MSCALE. By default (MSCALE = 1), LINPROG makes only a *row scaling*, which has the effect that the maximum norm of each scaled row becomes unity:

$$\max\{|a_{ij}| : j = 1, \ldots, n_s\} = 1. \tag{193}$$

The objective row is not scaled. Experiments show that this type of scaling is adequate for a major class of LP problems. Sometimes, however, it might be better to use *column scaling*. This is achieved in LINPROG by specifying MSCALE = 2. Then the maximum norm of each scaled column becomes unity:

$$\max\{|a_{ij}| : i = 1, \ldots, m\} = 1. \tag{194}$$

As a third possibility (MSCALE = 3) LINPROG can scale rows *and* columns, such that the scaled coefficients satisfy (193) as well as (194). A simple code for this purpose was found in Bartels *et al.* [18]. We use their method and compute the row and column scales, r_i and s_j, by the following algorithm (given in pseudo-PASCAL):

FOR $i := 1$ **TO** m **DO** $r_i := 0$;
FOR $j := 1$ **TO** n_s **DO**
  BEGIN
    $h := 0$;
    FOR $i := 1$ **TO** m **DO** $h := \max(h, |a_{ij}|)$;
    $s_j := 1/h$;
    FOR $i := 1$ **TO** m **DO** $r_i := \max(r_i, s_j|a_{ij}|)$
  END;

The matrix elements can now be normalized by r_i and s_j as follows:

FOR $j := 1$ **TO** n_s **DO**
  FOR $i := 1$ **TO** m **DO** $a_{ij} := s_j/r_i \times a_{ij}$;

It is easy to see that after this operation both (193) and (194) are fulfilled. In our experience MSCALE = 3 is a good choice for ill-conditioned and badly scaled LP problems, where MSCALE = 1 may be inadequate.

Of course, row scaling affects the right-hand side as well. Likewise, column scaling induces a scaling of the structural variables.

Other LP systems may offer more elaborate scaling procedures, though the benefit from them is sometimes questioned. In some systems it is possible to specify column scale numbers s_j directly in the input. A survey of scaling methods for LP is given by Tomlin [25].

It should be noticed that the LINPROG user can suppress the scaling completely by letting MSCALE = 0. This could be appropriate when the original formulation (1) is already well scaled.

LINPROG uses 9 fundamental tolerance parameters. The number of parameters as well as their default values are subject to possible future changes. Using the FORTRAN names, we list in Table 3 the present setting and use of the parameters. The default values were chosen after much trial work with our test problems;

Parameter	Value	Usage
BIG	$1.0\ 10^{+30}$	"Approximation" of $+\infty$
EPSCHC	$1.0\ 10^{-10}$	Reduced cost tolerance
EPSCHR	$1.0\ 10^{-10}$	CHUZR tolerance
EPSFEA	$1.0\ 10^{-10}$	Feasibility tolerance
EPSINA	$1.0\ 10^{-08}$	Tolerance for nonzero input coefficients
EPSLU	$1.0\ 10^{-13}$	Tolerance for new elements in eta vectors
EPSPIV	$1.0\ 10^{-09}$	Minimum pivot size in CHUZR
EPSRIN	$1.0\ 10^{-02}$	Minimum relative pivot size in inversion
ZERPIV	$1.0\ 10^{+00}$	Minimum pivot size in Phase 0

Table 3. Fundamental tolerance parameters in LINPROG

they apply to computers with (at least) 8 bytes working precision. The choice of tolerances should be suitable for a wide class of problem types. As explained in Section 3.2, the default settings can be modified by use of numerical keywords in the Control File.

The parameter BIG is a computer substitute for infinity. The other parameters are used to "kill" small numbers, to perturb feasibility ranges, resolve "ties", or prevent use of small pivots. We give a short account of each of them in the following.

The parameter EPSCHC is used in the CHUZC procedure for deciding when a reduced cost d_j is insignificant. We use a relative test:

$$|d_j| \leq \text{EPSCHC} \, \| \pi \|_1 \quad \Rightarrow \quad d_j \text{ insignificant}, \tag{195}$$

where π is the pricing vector defined in (13) (or (167) for Phase 1); $\| \cdot \|_1$ stands for the"1-norm", which is the sum of the absolute values of the vector components. The test (195) is also used to set insignificant dual activities to zero in the output (these are reduced costs for the slack variables.)

The parameter EPSCHR is used in CHUZR. It works by making a perturbation of the feasible range, thus permitting very small infeasibilities. Given a basic solution $\mathbf{x}_B = \beta = \{\beta_i\}$ $(i = 1, \ldots, m)$, LINPROG first computes the perturbation

$$\text{TOLCHR} = N(\beta) \, \text{EPSCHR}, \tag{196}$$

where N is a norm-like function defined by

$$N(\beta) = \frac{1}{\sqrt{m}} \max\{\| \beta \|_1, m\}, \tag{197}$$

and then replaces the feasibility test in (155) by

$$\ell_i - \text{TOLCHR} \leq \beta_i \leq u_i + \text{TOLCHR}. \tag{198}$$

The heuristic formula (197) was chosen after theoretical and practical considerations combined with numerical experiments. Experience shows that the $\sqrt{m}$ divisor, which also occurs in tests in other LP codes, e.g. [6], provides sensible feasibility tolerances TOLCHR over a wide span of matrix sizes and sparsities. Notice that N is not a genuine norm function, as $N(\mathbf{0}) = \sqrt{m} > 0$. We made this amendment to prevent TOLCHR to become exactly zero, should β be zero.

Consider now the important default case $u_i = +\infty$. As LINPROG internally sets $\ell_i = 0$, (198) reduces to

$$0 \leq \beta_i + \text{TOLCHR}; \tag{199}$$

in the ratio test (21) we therefore replace the numerator β_i by β_i + TOLCHR. This "perturbed" test has the advantage that should two ratios in (21) be equal, then priority is given to the larger pivot size $|\alpha_{iq}|$, thus enhancing the numerical stability.

In ill-conditioned or badly scaled LP problems the step length θ determined by CHUZR may be difficult to discern from zero, due to the competition of rounding errors with the tolerance TOLCHR. This could lead to bad decisions in CHUZR and even to cycling. To reduce the probability of this kind of cycling or stalling we monitor the object function regularly; if no progress is found, then TOLCHR is temporarily set to zero for a few iterations. Experience shows that such a disturbance is likely to break the cycle.

Another use of EPSCHR in CHUZR is as a tool for *tie breaking*. Often in simplex a candidate is selected from some minimum or maximum criterion, and when several candidates are equal, we must break the tie. This can usually be done arbitrarily, by taking the first candidate in the list. Sometimes we use a secondary criterion, say pivot size, as we did in the inversion procedure (Section 4.4.4). We have seen examples where CHUZR makes indefinitely alternating selections between θ_3 and θ_1 in (192). When a tie occurs, cycling is counteracted by a systematic favouring of one of the possibilities. We do this by replacing θ_3 in (192) with $\theta_3/(1 + \text{EPSCHR})$. (In other cases θ_3 was selected every time, such that the state of a nonbasic variable oscillated between its two bounds. This situation could be avoided by forbidding an immediate re-selection of θ_3.)

The use of EPSFEA is similar to EPSCHR. It is used to derive a tolerance

$$\text{TOLFEA} = N(\beta) \, \text{EPSFEA}, \tag{200}$$

which in turn is applied in the evaluation of the infeasibility indicators NINF and SINF (Section 4.6.3). We could have used a common parameter for TOLCHR and TOLFEA, but two parameters give more flexibility in monitoring the feasibility of the LP solutions.

The tolerance parameter EPSINA is used as a filter for small input elements in the constraint matrix, in the following simple way:

$$|a_{ij}| \leq \text{EPSINA} \quad \Rightarrow \quad a_{ij} := 0. \tag{201}$$

Such a device is useful when the input comes from a matrix generator.

The parameter EPSLU is used in the same way as a filter for small elements in the newly created eta vectors in the Forrest-Tomlin updating process. For the column vector (cf. (109)) we make the test

$$|v_i| \leq \text{EPSLU}\,|\overline{u}_{pp}| \quad \Rightarrow \quad v_i := 0. \tag{202}$$

For the row vector in (74) we make a similar test, only we do not multiply EPSLU by $|\overline{u}_{pp}|$ in that case.

The parameter EPSPIV is the minimum pivot size we are willing to accept in a simplex iteration. It is used in CHUZR and in Phase 0. It is also used in the elimination process during presolve (Section 4.7).

The threshold parameter EPSRIN corresponds to ϵ in Section 4.4.4 and serves, as described there, to ensure the numerical stability of our re-inversion when choosing pivots.

Finally ZERPIV is a threshold parameter for the pivot size in Phase 0. It corresponds to ϵ_0 in Section 4.6.2, cf. the test (152). Numerical experiments indicate that the value of ZERPIV should be rather high.

If we compare the Forrest-Tomlin scheme in LINPROG by the Bartels-Golub update procedure as used e.g. in [18], we must admit that inherently the latter is the more stable of the two. With the Forrest-Tomlin method we may sometimes get rather small pivots, and if this happens too often, the representation of $\mathbf{B}^{-1}$ becomes inaccurate. As a consequence of this, we monitor the basic solution for feasibility also in Phase 2 using the TOLFEA tolerance of (200). To save work, however, we check the solution only after a re-inversion and again by the end of Phase 2. Should unexpected infeasibilities then be detected, the computations are thrown back to Phase 1, where feasibility normally is restored after a few iterations. This "repair" procedure seems to work well in practice. The idea behind the strategy may be characterized as "cure is better than prevention".

Rounding errors set ultimate bounds on how large LP problems we can solve. "Large" means here both large dimensions of the constraint matrix and many nonzero elements. A positive impact of the rounding errors is, however, that they help resolve degeneracies and thereby avoid cycling in practical calculations.

4.6.6 Sparse-matrix and other programming techniques

As mentioned previously the application of sparse-matrix technique is essential for handling large LP problems. There are several ways to represent sparse matrices and vectors in a computer. Most of the techniques in LINPROG are based on the so-called "ordered lists". Only the presolve module and parts of the inversion module use "linked lists". A survey of sparse-matrix methods can be found in the book by Duff *et al.* [26].

The constraint matrix is conveniently stored by columns with row numbers associated with each non-zero element. Only the non-zeroes are stored, and there are no gaps in the lists. Thus the matrix is stored as a collection of packed sparse vectors. As the computer is supposed to have virtual memory, it is practical to let the lists be two arrays in the program. In addition we use a pointer array holding

the addresses of the last non-zeroes in each column. With these lists it is easy to perform the necessary operations involving the constraint matrix.

In the Forrest-Tomlin updating scheme we may also represent the column eta vectors by a string of packed sparse vectors, and similarly for the row eta vectors.

Such a simple compact representation is not possible when fill-in occurs as a result of an updating process. In LINPROG this happens during the re-inversion of the basis matrix. We described in Section 4.4.5 how we used double ordered lists with elbow room as temporary storage for the "Boolean" part of the inversion. In Section 4.7 we shall see that fill-in may also occur in the presolve module, where we prefer to use a double set of linked lists with coefficient values in the row list only.

Before LINPROG can set up its sparse representation of the constraint matrix, an internal matrix generator must interpret the contents of the Matrix File discussed in Section 3.3. We recall that each row was given an 8-character name in the ROWS Section. In the COLUMNS Section the column names were supplied successively, finishing one column before going to the next. Alongside, with each column name a set of pairs was given (row name, element value) corresponding to the nonzero coefficients in the column. The row names could come in any order within a column. When converting this structure to an LP matrix, it would be expensive to identify the row names in the COLUMNS Section by scanning the total input list of row names each time. Instead LINPROG starts to sort the row names in the ROWS Section input in alphabetic order. To do this the names are mapped into pairs of integers corresponding to character positions 1–4 and 5–8 via the ASCII standard collating sequence defining the bit patterns of the characters. The same sorting procedure is executed for each column in the COLUMNS Section input. This means that the row names in a column can be identified in a single merge-scan with the total sorted list of row names. The sorting algorithms applied in LINPROG are variants of the quicksort method of van Emden [27].

4.7 Presolve module

LINPROG is equipped with a so-called "presolve" or "reduction" module. This is able to decrease the effective size of the LP problem by recursive eliminations in the constraint matrix prior to the simplex optimization process. By this process redundant variables and constraints will be removed, and certain balance equations identified and eliminated.

The presolve facility is activated by using the keyword REDUCE (Chapter 3).

When REDUCE is on, LINPROG will print a short summary of the presolve statistics. This might look as follows:

```
====================== REPORT ON MATRIX REDUCTION ======================

SIZE OF UNREDUCED LP MATRIX: 1723 X  1639
SIZE OF   REDUCED LP MATRIX:  808 X   825

  SCANS   SINGLE-  IMPLIED  IMPLIED  IMPLIED   SIGN    ELIMINA-  FINAL
 IN ROW     TON     LOWER    UPPER    FIXED    REDUN-  TED TREE LENGTH
 SEARCH    ROWS    BOUNDS   BOUNDS   BOUNDS   DANCIES   EQNS.   OF LIST
    4       222       10       41      160       31      646      7456
========================================================================
```

DUMP/RESTART works properly together with REDUCE, provided the user takes care of restarting only REDUCE problems from REDUCE dumps. If not, a conflict in the matrix size may occur. But even with this precaution the restart will not always be successful. The size of the restart matrix after reduction must match the reduced dump matrix exactly. This is the case if the Matrix File is unchanged. If there are changes in some coefficients or in the right-hand side,

then there is a risk that the eliminations will proceed differently, resulting in a reduced matrix of a different size. Anyway, an incompatible restart matrix will be discovered by LINPROG, and a message will be given.

All vectors and matrices in LINPROG are physically reduced to smaller sizes (in contrast to e. g. the reduction technique proposed by Tomlin and Welch [28]). After the reduced system is optimized, a recreation procedure must therefore be invoked. In LINPROG this is implemented by taking advantage of the capability of the code to restart from any basic index set.

Our design of the presolve module was inspired by similar reduction methods discussed in the documentation [29] of the EFOM model, which is an important target for the application of LINPROG (see Chapter 8). Consequently LINPROG's presolve module is particularly efficient on EFOM applications.

It is a crucial part of our reduction process to identify the so-called nodal balance equations (also called tree equations or transfer equations). An equation of this type can be written

$$a_{ik}x_k + \sum_{j \neq k} a_{ij}x_j = 0, \tag{203}$$

where a_{ik} is positive and all the other coefficients are negative. More generally we may consider an equation

$$a_{ik}x_k + \sum_{j \neq k} a_{ij}x_j = b_i, \tag{204}$$

with the additional requirement that b_i is non-negative.

Such constraints permit a recursive elimination of the "tree variables" from the LP, where the positivity is guaranteed automatically after the elimination. A proper elimination order can decrease the volume of matrix fill-in and arithmetics during reduction significantly. We tried several strategies for the elimination order and identified four promising approaches:

1. minimal fixed Markowitz count (FM),
2. minimal dynamical Markowitz count (DM),
3. minimal fixed row count (FR), and
4. minimal dynamical row count (DR).

No obvious conclusion could be drawn about which strategy was uniformly the best. Apparently DM was slightly better overall than FM but also more complicated, so we decided to use FM in LINPROG. It is capable of keeping matrix fill-in and arithmetics at a low level during the reduction process, using fairly simple coding.

The mechanism in LINPROG's presolve algorithm consists in consecutive locating of singleton rows, redundant inequalities, and nodal balance equations, respectively:

- **Singleton rows**: We make a recursive scan for such rows in the LP matrix, using a certain number of search passes. For each singleton row we decide whether the row corresponds to an implied lower or upper bound, or to a redundant restriction; in all these cases the row can be eliminated. If the row implies a fixed value of a variable, the constraint and the variable are both eliminated. Finally a singleton row may be infeasible.
- **Redundant inequalities**: Based on a sign test the next scan tries to detect and eliminate redundant inequalities.
- **Nodal balance equations**: Finally, equations of the type (204) are identified and sorted according to their Markowitz counts, giving a proper order for elimination of the equations and corresponding pivot variables.

The sparse-matrix eliminations are facilitated by a double set of linked lists with coefficient values only in the row list.

Typically the presolve module was able to decrease the running time for EFOM problems by almost 50%.

An additional gain in using the presolve facility is that many infeasible LP problems can be diagnosed before the simplex procedure comes into work.

5 Computer environment

This chapter gives informations and hints about running LINPROG on various computer systems. Information on the organisation of the source code is found in Appendix B, while Appendix C describes the installation of LINPROG.

5.1 Computer platforms for LINPROG

LINPROG is written in standard FORTRAN 77, and may thus be implemented at all computer systems supporting FORTRAN. The list of computers, where LINPROG has been made to run, covers the following:

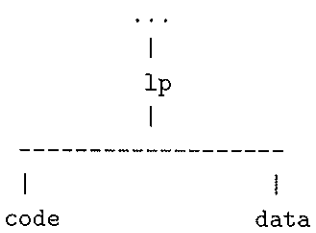
- DEC (VAX, Alpha, Ultrix)
- UNISYS A-series
- Apollo DN10000
- HP9000
- IBM/R6000
- SUN 4/75 Sparc
- Silicon Graphics Indigo
- Convex 220
- PCs using 386 and 486 based FORTRAN compilers under DOS:
  - Lahey F77L-EM/32 and LF90
  - FTN77/386 from Salford University
- PCs using Pentium based FORTRAN compiler under OS/2: WATCOM

5.2 How to run LINPROG jobs

In the following we propose a very simple way of organising the execution of LINPROG jobs. Some users might need more elaborate structures that also includes the LINPROG model environment (see Chapter 7). We shall restrict ourselves to three main types of platforms: PC, Unix, and DEC VMS (Alpha or VAX).

All these systems have hierarchical, tree-structured, file systems, where each file can be found within a directory. We can therefore use a common organisation of the LINPROG software covering all three cases. The structure proposed below is only meant as a guideline. You may wish to make local modifications to suit your own demands, which should be fairly easy to do.

A simple LINPROG installation may have a directory say "lp" with subdirectories "code" and "data", as shown below:



In the "code" subdirectory you should have the executable LINPROG code file; its name is normally "linprog.exe" or just "linprog". The subdirectory "data" contains Matrix Files (Section 3.3) for problems to be solved with LINPROG.

The "lp" directory itself is the master directory for controlling the LINPROG runnings and could for example be placed beneath the users's root directory. In "lp" there should be a command file "linrun" (possibly with the extension .bat or .com). This command file has (at least) one parameter, which is the name of the Matrix File (without .dat). For example, if you want to run LINPROG for solving an LP problem whose Matrix File is say "myproblm.dat" residing in the "data" directory, you should edit the Control File (Section 3.2) to specify proper LINPROG options, and then issue the command "linrun myproblm" (or "@linrun myproblm" for VMS). This causes LINPROG to solve the LP in question, and the output file, which here will be named "myproblm.lis", will afterwards be placed in the "lp" directory.

This general structure covers all three systems. However, differences are found in the "linrun" command file. In the following we give examples of individual command files for a couple of simple LINPROG jobs. This is done separately for PC-DOS/Unix and for VMS.

5.3 LINPROG running under DOS or Unix

We first present a simple version of the command file. For DOS we have a BAT file "linrun.bat", which may look as follows:

```
rem run linprog with data set %1.dat
copy .\data\%1.dat matrix.dat
.\code\linprog
copy result.lis %1.lis
```

while the corresponding Unix command file reads:

```
# run linprog with data set $1.dat
cp ./data/$1.dat matrix.dat
./code/linprog
copy result.lis $1.lis
```

Your Control File "control.dat" is supposed to reside in the directory "lp" and may look as follows:

```
solution
exec
pend
```

You can now type for example "linrun dietprob", assuming that you have a Matrix File "dietprob.dat" in your subdirectory "data". As a result LINPROG will run and produce the output file "dietprob.lis" with the contents discussed in Section 3.4. It will be placed in the "lp" directory.

A slightly more complicated command file is needed for a restart run of LINPROG. Below we show only the DOS BAT file:

```
rem restart from the dump file dump.dat
copy dump.dat restart.dat
rem run linprog with data set %1.dat
copy .\data\%1.dat matrix.dat
.\code\linprog
copy result.lis %1.lis
```

Here we assume that one of your previous LINPROG runs has left a dump file "dump.dat" in the directory "lp". The BAT file copies this to the restart file "restart.dat", and LINPROG is now ready to execute the restart job. Your Control File should in this case contain a restart instruction:

```
restart
solution
exec
pend
```

Note that for the PC and Unix versions the LINPROG files have the fixed names given in Table 2 in Section 3.1.

5.4 LINPROG running on DEC VMS

We shall next give examples of command files for controlling the execution of LINPROG on a Digital Alpha AXP or VAX computer under VMS. The command files have normally the extension ".com", and they are written in the so-called Digital Control Language (DCL). With DCL it is possible to embed the LINPROG Control File (unit 19, cf. Table 2 in Section 3.1) in the command file. (VMS FORTRAN files need not be given names within the program.) We assume a directory structure as outlined in Section 5.2. As a first and very simple example we show the following "linrun.com" file:

```
$define/user for019 sys$input
$define/user for002 [.data]'p1'.dat
$define/user for006 'p1'.lis
$run [.code]linprog
   solution
   exec
   pend
$exit
```

This command file takes the name of the matrix file as a parameter. To process it in interactive mode you just type @LINPROG <filename>, where <filename> is the name of the Matrix File you intend to use. (Alternatively, you may submit the job for batch processing by the SUBMIT command.)

Next we give a little more sophisticated example, where a restart file is read and a dump file is produced and converted into a restart file for a later run. Also, a communication file is produced, and afterwards this is read by the report-writer program REPORT discussed in Section 3.6 and shown in Appendix A:

```
$define/user for019 sys$input
$define/user for002 [.data]'p1'.dat
$define/user for006 'p1'.lis
$define/user for015 restart.dat
```

```
$define/user for016 dump.dat
$define/user for020 binout.dat
$run [.code]linprog
   reduce
   restart
   dump
   binout
   exec
   pend
$rename dump.dat restart.dat
$define/user for001 binout.dat
$define/user for002 report.out
$run [.code]report
$exit
```

There is an important restriction you may face as a VMS LINPROG user: the "pagefile quota" for your username should have a fairly high value, and this must be set by your VMS system manager whom you are advised to consult.

Moreover, if you plan to use VMS LINPROG for solving large problems, it may be a good idea to get your maximum working set "WSextent" raised to a higher value than the default, if this is possible. This will give a considerable reduction in the number of "pagefaults" and thereby also of the CPU-time. Again, you should consult your system manager.

6 Test of LINPROG

LINPROG has undergone an extensive testing with hundreds of sample problems. These span in size from LP matrices with only a couple of rows and columns to about 16000 × 16000 for the largest. In Sections 6.1 and 6.2 we shall present the results of a systematic application of test examples from our two main sources NETLIB and EFOM, while we in Section 6.3 test selected NETLIB and EFOM cases across different computer platforms.

However we have also collected many test problems from other sources. For example, we have taken small examples from a textbook in numerical analysis by Fröberg [30], from Cohn's book on linear algebra [31], from the books on LP by Murtagh [8] and by Nazareth [4], from Garfinkel and Nemhauser's book on integer programming [23], and from the user guides for two other LP systems, MPSX [5] and MINOS [6]. In Sections 8.1.1 and 8.1.2 we describe some LP applications at Risø, which have supplied further test problems for LINPROG. Of these Weber's test cases are small but quite dense and may present some numerical difficulties, while Munksgaard Pedersen's cases [32] are small to medium with a maximum of about 1400 rows. Finally we have constructed problems to test certain special features. An example is the solution of 200 equations in 200 unknowns with a random sparsity pattern and random values of the nonzeroes.

6.1 NETLIB test

Since 1990 we have used the collection of LP test cases from the NETLIB service [2, 33]. These test cases are recognized worldwide as benchmark objects for solution of LP problems. In the meantime the collection has been enlarged from 65 to 94 problems, with some quite difficult examples among the newcomers. The NETLIB cases proved to be of utmost importance for the consolidation of LINPROG.

6.1.1 Models behind the test examples

The "NETLIB index from lp/data" [33] contains information about the origin of the various test examples. We quote from this in annotated form:

BLEND is a variant of the oil refinery problem in Murtagh's book [8].

BOEING1 and BOEING2 have to do with flap settings on aircraft for economical operations.

D6CUBE solves a problem of finding the minimum cardinality of triangulations of the 6-dimensional cube.

DFL001 is a real-world airline schedule planning (fleet assignment) problem.

FINNIS is a Harwell model for the selection of alternative fuel types.

FIT1D, FIT1P, FIT2D, FIT2P originate from a model for fitting linear inequalities to data.

GFRD-PNC and SIERRA come from network programming for forest planning.

GREENBEA and GREENBEB are both related to a large refinery model.

LOTFI is a semi-real world problem involving audit staff scheduling.

MAROS is an industrial production/allocation model. MAROS-R7 is an image-restoration problem done via a goal programming approach. MODSZK1 is a real-life problem from a multi-sector economic planning model of the input/output type.

PEROLD, PILOT, PILOT.JA, PILOT.WE, PILOT4, PILOT87, PILOTNOV are linearized forms of quadratic programs.

STOCFOR1,2,3 are stochastic forestry problems.

TRUSS is a problem of minimizing the weight of a certain structure.

6.1.2 Summary of LINPROG test results

The 94 NETLIB cases from January 1995 were solved with LINPROG using an HP 9000/755 workstation. To save space, we show in Table 4 only the output summary line (cf. Section 3.4) produced by LINPROG for each problem (the * is explained in Section 6.1.3):

Table 4. NETLIB test cases solved with LINPROG

CASE	ROWS	COLS	ELEM	PH.0	PH.1	ITNS	VIOL	OPTIMUM	CPU-SEC
25FV47	822	1571	11127	5	1076	4321	4.1E-12	5.5018458882867E+03	38.08
80BAU3B	2263	9799	29063P	0	936	5796	2.6E-12	9.8722419240909E+05	132.03
ADLITTLE	57	97	465	4	11	112	2.3E-13	2.2549496316238E+05	.09
AFIRO	28	32	88	0	4	11	1.8E-14	-4.6475314285714E+02	.02
AGG	489	163	2541	0	39	86	2.3E-10	-3.5991767286576E+07	.48
AGG2	517	302	4515	0	30	112	5.8E-11	-2.0239252355977E+07	.79
AGG3	517	302	4531	0	22	118	2.9E-11	1.0312115935089E+07	.81
BANDM	306	472	2659P	0	115	317	5.4E-12	-1.5862801845012E+02	.93
BEACONFD	174	262	3476	0	0	32	4.2E-11	3.3592485807200E+04	.28
BLEND	75	83	521	0	0	69	5.8E-12	-3.0812149845815E+01	.09
BNL1	644	1175	6129P	8	2744	3129	8.4E-13	1.9776295615229E+03	12.68
BNL2	2325	3489	16124P	0	1452	4799	3.4E-13	1.8112365403585E+03	45.30
BOEING1	352	384	3865	0	132	835	2.3E-13	-3.3521356750713E+02	2.36
BOEING2	167	143	1339	0	105	176	4.4E-13	-3.1501872801520E+02	.32
BORE3D	234	315	1525P	0	33	76	1.8E-12	1.3730803942085E+03	.19
BRANDY	221	249	2150P	37	166	361	8.0E-13	1.5185098964881E+03	.67
CAPRI	272	353	1786P	39	150	318	7.1E-13	2.6900129137682E+03	.65
CYCLE	1904	2857	21322P	4	0	792	1.1E-11	-5.2263930248941E+00	9.28
CZPROB	930	3523	14173P	0	698	1426	7.4E-13	2.1851966988566E+06	9.48
D2Q06C	2172	5167	35674P	134	1342	26608	5.0E-10	1.2278421081419E+05	689.16
D6CUBE	416	6184	43888	404	1657	20997	6.5E-12	3.1549166666667E+02	411.12
DEGEN2	445	534	4449	139	536	1415	4.9E-15	-1.4351780000000E+03	6.70
DEGEN3	1504	1818	26230	361	3261	7019	2.7E-14	-9.8729400000000E+02	206.43
DFL001	607212230		41873P25011.9E5835597				1.3E-12	1.1266397522847E+07	187839.4
E226	224	282	2767P	0	46	402	3.6E-14	-1.1638929066371E+01	.90
ETAMACRO	401	688	2489P	17	396	821	2.2E-13	-7.5571523276104E+02	2.08
FFFFF800	525	854	6235P	0	163	339	5.8E-11	5.5567956481750E+05	1.47
FINNIS	498	614	2714P	0	219	781	9.7E-13	1.7279106559561E+05	2.10
FIT1D	25	1026	14430	0	0	1006	3.6E-12	-9.1463780924209E+03	4.68
FIT1P	628	1677	10894	0	0	642	8.5E-14	9.1463780924209E+03	19.83
FIT2D	2610500138018			0	0	12155	5.4E-11	-6.8464293293832E+04	544.33
FIT2P	300113525	60784		0	0	10602	5.7E-14	6.8464293293832E+04	6343.26
FORPLAN	162	421	4916	0	62	175	9.3E-10	-6.6421896127220E+02	.67
GANGES	1310	1681	7021P	12	7	474	1.5E-11	-1.0958573612928E+05	2.74
GFRD-PNC	617	1092	3467P	0	128	490	3.6E-12	6.9022359995488E+06	1.50
GREENBEA	2393	5405	31499	2	4993	14157	3.1E-08	-7.2555238968172E+07	308.32
GREENBEB	2393	5405	31499	2	3755	8928	3.3E-11	-4.3022602581358E+06	198.89
GROW15	301	645	5665	0	0	539	3.3E-06	-1.0687094129358E+08	3.29
GROW22	441	946	8318	0	0	925	1.3E-08	-1.6083433648256E+08	7.33
GROW7	141	301	2633	0	0	193	3.0E-09	-4.7787811814711E+07	.66
ISRAEL	175	142	2358	0	0	330	1.0E-10	-8.9664482186305E+05	.79
KB2	44	41	291	0	0	48	4.5E-12	-1.7499001299062E+03	.05
LOTFI	154	308	1086P	0	25	124	9.3E-10	-2.5264706061880E+01	.19
MAROS	847	1443	10006P	0	498	1411	6.8E-10	-5.8063743701126E+04	8.88
MAROS-R7	3137	9408151120P2152			972	5161	3.0E-10	1.4971851664796E+06	255.70
*MODSZK1	688	1620	4158	0	0	558	1.2E-10	3.2061972906431E+02	3.05
NESM	663	2923	13988	0	1065	4698	1.7E-11	1.4076037591001E+07	36.21
*PEROLD	626	1376	6026	0	1490	6414	1.5E-05	-9.3807554218341E+03	48.94
*PILOT	1442	3652	43220	0	10734	21192	3.2E-08	-5.5748972927440E+02	2635.99
*PILOT.JA	941	1988	14706	0	2135	18848	1.5E-05	-6.1131372596992E+03	252.00
*PILOT.WE	723	2789	9218	0	802	12193	2.0E-05	-2.7201075859066E+06	127.22
*PILOT4	411	1000	5145	0	217	1943	2.5E-07	-2.5811392612812E+03	12.08

CASE	ROWS	COLS	ELEM	PH.0	PH.1	ITNS	VIOL	OPTIMUM	CPU-SEC
*PILOT87	2031	4883	73804	0	13891	59070	3.1E-09	3.0171035933700E+0234495.57	
*PILOTNOV	976	2172	13129	0	3224	4092	1.5E-05	-4.4972761882192E+03	48.10
RECIPE	92	180	752	3	7	27	.0E+00	-2.6661600000000E+02	.08
SC105	106	103	281	0	0	65	4.5E-13	-5.2202061211707E+01	.07
SC205	206	203	552	0	0	154	1.6E-12	-5.2202061211707E+01	.26
SC50A	51	48	131	0	0	32	8.5E-14	-6.4575077058564E+01	.02
SC50B	51	48	119	0	0	31	5.7E-14	-7.0000000000000E+01	.01
SCAGR25	472	500	2029P	0	396	573	3.6E-12	-1.4753433060769E+07	1.33
SCAGR7	130	140	553P	0	74	122	1.4E-12	-2.3313898243310E+06	.12
SCFXM1	331	457	2612P	8	131	265	1.6E-12	1.8416759028349E+04	.69
SCFXM2	661	914	5229P	16	355	606	2.8E-12	3.6660261564999E+04	2.53
SCFXM3	991	1371	7846P	24	588	1017	3.9E-12	5.4901254549751E+04	5.80
SCORPION	389	358	1708P	0	58	93	3.3E-16	1.8781248227381E+03	.25
SCRS8	491	1169	4029P	0	5	403	2.8E-14	9.0429695380079E+02	1.36
SCSD1	78	760	3148	0	48	244	3.0E-16	8.6666666743334E+00	.58
SCSD6	148	1350	5666	0	115	562	1.1E-15	5.0500000078262E+01	1.99
SCSD8	398	2750	11334	0	406	1102	1.9E-15	9.0499999992546E+02	7.18
SCTAP1	301	480	2052	0	152	210	7.1E-15	1.4122500000000E+03	.62
SCTAP2	1091	1880	8124	0	216	550	1.4E-14	1.7248071428571E+03	4.39
SCTAP3	1481	2480	10734	0	277	699	1.4E-14	1.4240000000000E+03	7.33
SEBA	516	1028	4874	0	78	113	1.9E-12	1.5711600000000E+04	1.26
SHARE1B	118	225	1182P	60	95	262	6.0E-10	-7.6589318579186E+04	.33
SHARE2B	97	79	730P	1	81	135	2.3E-13	-4.1573224074142E+02	.15
SHELL	537	1775	4900P	280	109	584	.0E+00	1.2088253460000E+09	1.54
SHIP04L	403	2118	8450P	0	427	529	1.2E-14	1.7933245379704E+06	2.76
SHIP04S	403	1458	5810P	0	10	158	2.9E-14	1.7987147004454E+06	.94
SHIP08L	779	4283	17085P	0	296	798	4.4E-14	1.9090552113891E+06	8.37
SHIP08S	779	2387	9501P	0	16	298	4.2E-14	1.9200982105346E+06	2.26
SHIP12L	1152	5427	21597P	0	1175	1589	4.0E-14	1.4701879193293E+06	20.21
SHIP12S	1152	2763	10941P	0	462	711	4.8E-14	1.4892361344061E+06	4.61
SIERRA	1228	2036	9252	505	143	919	4.5E-13	1.5394362183632E+07	5.15
STAIR	357	467	3857	98	328	710	2.1E-10	-2.5126695119296E+02	3.94
STANDATA	360	1075	3038P	0	17	25	2.3E-13	1.2576995000000E+03	.41
STANDMPS	468	1075	3686P	0	97	175	1.0E-13	1.4060175000000E+03	.85
STOCFOR1	118	111	474P	0	16	34	4.5E-13	-4.1131976219436E+04	.06
STOCFOR2	2158	2031	9492P	0	288	855	7.3E-12	-3.9024408537882E+04	8.24
*STOCFOR3	1667615695		74004P	0	796	9967	7.7E-12	-3.9976783943650E+04	1043.66
TRUSS	1001	8806	36642	909	632	10518	1.6E-13	4.5881584718562E+05	201.19
*TUFF	334	587	4523P	0	81	163	9.1E-13	2.9214776509361E-01	.94
VTP.BASE	199	203	914P	0	202	242	2.9E-11	1.2983146246136E+05	.47
WOOD1P	245	2594	70216	241	80	601	2.0E-13	1.4429024115734E+00	13.80
WOODW	1099	8405	37478	0	396	1453	1.1E-13	1.3044763330842E+00	34.86

6.1.3 Comments on test results

We have used the presolve facility of LINPROG whenever the test problems could benefit from it. These cases are marked with a P (in column position 25). Apart from this, most NETLIB problems were solved with the default settings of LINPROG's parameters and options.

However, those problems whose summary lines are marked with an asterisk were solved with the scaling option MSCALE = 3. (Recall from Sections 3.2 and 4.6.5 that the numerical keyword MSCALE in the Control File governs the scaling of the LP matrix, such that the default option MSCALE = 1 gives *row scaling* while MSCALE = 3 gives *row and column scaling.*) Experience from NETLIB and other tests shows that proper scaling is very important for the correct functioning of LINPROG. This is because the set of program tolerances (Section 4.6.5) depends on a good scaling of the constraint matrix. Badly scaled problems may give rise to convergence problems and even result in a LINPROG error stop (NO SIMPLEX PROGRESS), meaning (numerical) cycling or stalling. In some cases we have seen

new infeasibilities continually showing up even in phase 2. Problems of this kind were noticed with some of the PILOT* cases, with STOCFOR3, and with TUFF. Bad scaling may also arise from use of presolve, as it does with GREENBEA and GREENBEB. The very degenerate case MODSZK1 gives a structurally singular basis and a subsequent inversion break-down, when MSCALE = 1 is attempted. In our experience MSCALE = 3 is a good option to use for the difficult NETLIB cases as well as for other troublesome problems.

The case MAROS-R7 presented another kind of difficulty, which indeed called for a modification of LINPROG: After the "tree balance equations" are eliminated by the presolve module (Section 4.7), the CRASH procedure (Section 4.6.2) will produce a triangular initial basis matrix $\mathbf{B}$ (slack-free in this example), but the subsequent calculation of the modified right-hand side $\mathbf{B}^{-1}\mathbf{b}$ would break down with an overflow. In the new version of LINPROG (9501) we circumvent this difficulty by monitoring the growth when computing $\mathbf{B}^{-1}\mathbf{b}$; if necessary we discard $\mathbf{B}$ and make instead a re-crashing, this time using an all-slack basis.

As seen from the list we could solve all the NETLIB cases, though some of them were very hard indeed. In particular DFL001 ran for ages before the optimum was reached. Such a problem would probably benefit from a more sophisticated pricing strategy, see for example Bixby *et al.* [22].

We have generally found quite good agreement with the MINOS results given in the "NETLIB index from lp/data" [1]. As stated there, R. Bixby observed slight differences for some cases, when comparing with values from his CPLEX code. In Table 5 we list Bixby's CPLEX optimal values together with our LINPROG results for these problems:

Table 5. Selected NETLIB cases solved with CPLEX and LINPROG

PROBLEM	CPLEX	LINPROG	REL.DIF.
80BAU3B	9.8722419241E+05	9.8722419241E+05	0.00E+00
BNL1	1.9776295615E+03	1.9776295615E+03	0.00E+00
DFL001	1.1266396047E+07	1.1266397523E+07	1.31E-07
ETAMACRO	-7.5571523337E+02	-7.5571523276E+02	8.07E-10
FFFFF800	5.5567956482E+05	5.5567956482E+05	0.00E+00
FORPLAN	-6.6421896127E+02	-6.6421896127E+02	0.00E+00
GANGES	-1.0958573613E+05	-1.0958573613E+05	0.00E+00
GREENBEA	-7.2555248130E+07	-7.2555238968E+07	1.26E-07
GREENBEB	-4.3022602612E+06	-4.3022602581E+06	7.21E-10
NESM	1.4076036488E+07	1.4076037591E+07	7.84E-08
PEROLD	-9.3807552782E+03	-9.3807554218E+03	1.53E-08
PILOT	-5.5748972928E+02	-5.5748972927E+02	1.79E-11
PILOT.JA	-6.1131364656E+03	-6.1131372597E+03	1.30E-07
PILOT.WE	-2.7201075328E+06	-2.7201075859E+06	1.95E-08
PILOT4	-2.5811392589E+03	-2.5811392613E+03	9.30E-10
SCAGR7	-2.3313898243E+06	-2.3313898243E+06	0.00E+00
SCRS8	9.0429695380E+02	9.0429695380E+02	0.00E+00
SCSD6	5.0500000077E+01	5.0500000078E+01	1.98E-11
STOCFOR3	-3.9976785944E+04	-3.9976783944E+04	5.00E-08

We notice a good agreement in the optimal values between CPLEX and LINPROG.

We have also tried LINPROG with the NETLIB set of infeasible LP testproblems collected by John W. Chinneck. All these 29 problems were correctly flagged as infeasible by LINPROG. In this test we used the default options. In a few cases the infeasibilities were detected before the onset of the simplex procedure.

[1] The discrepancy for E226 is only apparent: the values differ with the constant objective coefficient -7.113

6.2 EFOM test

In Chapter 7 we shall describe the EFOM computer modelling package [29] developed by the European Commission and intended for nationwide optimization of energy systems. Over the years EFOM has given much impetus to the development and testing of LINPROG.

Here we have selected a few EFOM cases as typical representatives for problems of different sizes. Like the NETLIB cases we solved the EFOM test problems with an HP 9000/755 workstation. In Table 6 we list only the LINPROG summary lines produced:

Table 6. EFOM test cases solved with LINPROG

CASE	ROWS	COLS	ELEM	PH.0	PH.1	ITNS	VIOL	OPTIMUM	CPU-SEC
DEMO428	428	394	1616P	0	165	374	3.5E-10	3.5874287430892E+12	.54
DEMO515	515	375	2855P	0	174	311	9.3E-10	3.4414928053428E+12	.51
DK12E1	1768	1639	8516P	0	249	959	9.2E-14	1.0022955086137E+08	5.16
EFFN00-A	5645	5962	36521P	95	1290	5650	4.0E-12	8.0397982342634E+08	117.33
UREF00-A	6946	6430	46365P	147	2720	8189	1.0E-11	3.0166537667378E+05	173.02
UREF01	8500	7853	58405P	162	3868	12358	2.8E-10	3.6930507702368E+05	336.45
VTTTAXA	1054510302		68421P		012490	84062	3.6E-12	1.6154776538953E+06	4658.55

The two problems DEMO428 and DEMO515 are small demonstration cases related to a sample country "Demoland". DK12E1 concerns a study of the Nordic electricity system (May 1993). The problem EFFN00-A has been used extensively as a LINPROG benchmark case. It occurred in an study of 30% emissions reduction of NO_x (February 1990). The two following cases come from the UK energy system: UREF00-A has 5 time periods, while UREF01 has 6 periods. Finally the large case VTTTAXA was communicated to us by VTT in Finland. It comes from a modeling of the total Finnish energy system.

As seen we have used the presolve facility of LINPROG in all the EFOM test cases. Experience shows that this saves nearly 50% computer time. Standard scaling (MSCALE = 1) was used throughout; generally this seems to be a better choice than MSCALE = 3 for EFOM cases.

6.3 Test across computer platforms

We have also made tests with LINPROG running on different computer platforms for three selected test cases from Sections 6.1 and 6.2, 8. The results are given in Tables 7, 8 and 9:

Table 7. NETLIB case CAPRI solved with LINPROG on different computers

Computer platform	FORTRAN compiler	LINPROG Version	Max rows	Time cpu-sec
DEC AlphaServer 1000 4/233	VMS Fortran V6.1-1	9501		0.48
HP 9000/755 Workstation	f77 -O	9501		0.65
SGI Power Challenge R8000	f77 -O1	9501		1.24
DEC VAX-4600 VMS	VMS Fortran	9501		1.32
DEC 5000/133 Ultrix 4.3	f77 -O	9501		2.57
SUN 4/75 Sparc Station 2	f77 -O	9501		3.40
SUN 4/75 Sparc Station 2	f77 -O	9311		3.50
Convex 220	fc -O1	9501		4.92
PC 486-33 256k cache 4 MB RAM	Salford FTN77/386 2.51	9201		19.51
PC 486-33 256k cache 4 MB RAM	Lahey F77L-EM/32 5.01	9201	12209	50.00
PC 486-33 256k cache 4 MB RAM	Lahey F77L-EM/32 5.01	9201	2047	17.00
PC 486-33 256k cache 4 MB RAM	Lahey F77L-EM/32 5.01	9201	511	9.00

Computer platform	FORTRAN compiler	LINPROG Version	Max rows	Time cpu-sec
PC NCR 386SX-20 4 MB RAM ITT Xc87co	Lahey F77L-EM/32 5.10	9201	12209	96.00
PC NCR 386SX-20 4 MB RAM ITT Xc87co	Lahey F77L-EM/32 5.10	9201	2048	68.00
PC NCR 386SX-20 4 MB RAM ITT Xc87co	Lahey F77L-EM/32 5.10	9201	550	49.00
PC NCR 386SX-20 4 MB RAM ITT Xc87co	Salford FTN77/386 2.51	9201		42.75
PC 486-33 256k cache 8 MB RAM	Lahey F77L-EM/32 5.20	9501	12209	9.66
PC 486-33 256k cache 8 MB RAM	Lahey F77L-EM/32 5.20	9501	2048	7.58
PC 486-33 256k cache 8 MB RAM	Lahey F77L-EM/32 5.20	9501	550	6.15
PC NCR 3230 486-66 8 MB RAM	Lahey F77L-EM/32 5.10	9411	16676	14.72
PC NCR 3230 486-66 8 MB RAM	Lahey F77L-EM/32 5.10	9411	12209	10.98
PC NCR 3230 486-66 8 MB RAM	Lahey F77L-EM/32 5.10	9411	2048	5.05
PC NCR 3230 486-66 8 MB RAM	Lahey F77L-EM/32 5.10	9411	550	4.72
PC Compaq LTE Lite 4/25E 8 MB RAM	Lahey F77L-EM/32 5.10	9411	12209	26.03
PC Olivetti M380-XP1 Cyrix copr.	Salford FTN77/386 2.61	9501		32.69
PC Toshiba T2200SX 6MB RAM,IIT Xc87.	Lahey F77L-EM/32 5.20.	9501	2047	64.27

Table 8. EFOM case DK12E1 solved with LINPROG on different computers

Computer platform	FORTRAN compiler	LINPROG Version	Max rows	Time cpu-sec
DEC AlphaServer 1000 4/233	VMS Fortran V6.1-1	9501		3.48
HP 9000/755 Workstation	f77 -O	9501		5.16
SGI Power Challenge R8000	f77 -O1	9501		10.19
DEC VAX-4600 VMS	VMS Fortran	9501		11.33
DEC VAX-8700 VMS	VMS Fortran	9201		64.50
DEC 5000/133 Ultrix 4.3	f77 -O	9501		21.89
SUN 4/75 Sparc Station 2	f77 -O	9501		26.56
Convex 220	fc -O1	9501		38.48
PC 486-33 256k cache 8 MB RAM	Lahey F77L-EM/32 5.01	9201	12209	98.00
PC 486-33 256k cache 8 MB RAM	Lahey F77L-EM/32 5.10	9201	12209	121.00
PC 486-33 256k cache 8 MB RAM	Lahey F77L-EM/32 5.10	9201	2048	88.70
PC NCR 386SX-20 4 MB RAM ITT Xc87co	Salford FTN77/386 2.51	9201		504.40
PC NCR 3230 486-66 8 MB RAM	Lahey F77L-EM/32 5.20	9501	16676	74.42
PC NCR 3230 486-66 8 MB RAM	Lahey F77L-EM/32 5.20	9501	12209	67.99
PC NCR 3230 486-66 8 MB RAM	Lahey F77L-EM/32 5.20	9501	2048	59.75
PC NCR 3230 486-66 8 MB RAM	Salford FTN77/386 2.51	9311		88.30
PC Compaq LTE Lite 4/25E 8 MB RAM	Lahey F77L-EM/32 5.10	9311	12209	118.04
PC Compaq LTE Lite 4/25E 8 MB RAM	F77L-EM/32 5.10	9411	12209	107.21
PC Toshiba T2200SX 6MB RAM,IIT Xc87.	Lahey F77L-EM/32 5.20.	9501	2047	470.00

Table 9. EFOM case EFFN00-A solved with LINPROG on different computers

Computer platform	FORTRAN compiler	LINPROG Version	Max rows	Time cpu-sec
DEC AlphaServer 1000 4/233	VMS Fortran V6.1-1	9501		68.89
IBM POWER2/590 (66 MHz) R6000 RISC	AIX	9311		102.55
IBM POWER2/590 (66 MHz) R6000 RISC	AIX	9310		123.36
HP 9000/735 Workstation	f77 -O	9311		135.59
HP 9000/755 Workstation	f77 -O	9501		117.33
HP 9000/750 Workstation	f77 -O	9201		197.76
SGI Power Challenge R8000	f77 -O1	9501		195.13
SGI 33 MHz Workstation	f77 -O	9311		547.65
DEC 3000 Alpha AXP/500 150 MHz) VMS	VMS Fortran	9201		157.46
DEC VAX-4600 VMS	VMS Fortran	9201		301.05
DEC VAX-4600 VMS	VMS Fortran	9501		269.20
DEC VAX-8700 VMS	VMS Fortran	9201		1540.02
DEC 5000/133 Ultrix 4.3	f77 -O	9501		562.29
SUN 4/75 Sparc Station 2	f77 -O	9501		538.84
Convex 220	fc -O1	9501		762.03
Convex 220	fc -O1	9201		1116.00
Apollo DN10000	f77 -O	9201		960.66

Computer platform	FORTRAN compiler	LINPROG Version	Max rows	Time cpu-sec
PC Compaq Deskpro 5/60M 15 MB RAM	Lahey F77L-EM/32 5.10	9311		507.12
PC Olivetti M380-XP1 Cyrix copr.	Salford FTN77/386 2.61	9201		6787.91
PC Olivetti M380-XP1 Cyrix copr.	Salford FTN77/386 2.61	9501		5567.09
PC IBM PS/2 Model 80	Salford FTN77/386 2.51	9201		9982.58
PC JAI 843 486-33 19 MB RAM	Lahey F77L-EM/32 5.10	9201		1635.80
PC JAI 843 486-33 19 MB RAM	Lahey F77L-EM/32 5.20	9501		1218.64
PC JAI 843 486-33 19 MB RAM	Lahey LF90 1.00 -02	9501		1010.30

The presolve option REDUCE was used throughout.

All the LINPROG runs in the three test suites resulted in correct values of the objective functions; these are therefore not printed in the tables.

The LINPROG version is given in the format "yymm". The "Max rows" column gives the declared maximum number of rows MR for the specific LINPROG implementation (cf. Appendices B and C). MR is only one of several bounds which together define the maximum problem size that can be handled by LINPROG. It is only stated for the Lahey F77L-EM32 PC compiler, where it has a major effect on the performance.

With the multitude of different computer systems involved, these tests have contributed to establish a certification of the correctness and portability of the LINPROG software.

The tables also reflect a certain evolution in the overall computing power in the time span covered. This pertains to the hardware as well as the software, i.e. the efficiency of the FORTRAN compilers and of LINPROG itself.

7 Model environment for LINPROG

Application of linear programming and other mathematical optimization techniques for large problems will require a model environment of the type shown in Table 10.

Table 10. The modular structure of an optimization model

General structure	EFOM model
Database	Energy network Techno-economic data Capacity data
Interface	Matrix generator
Optimizer	Linear programming software
Report writer	Energy system cost Capacity of new equipment Energy system cost

The database may be common for several modelling or statistical purposes. The key module of the particular model is the interface, in which these data are prepared for a general purpose mathematical programming solver, or merely the brain and intuition of the user. Finally, the output and the model assumptions are presented by a report writer.

7.1 The EFOM environment

The Energy Flow Optimization model, EFOM [29] describes the energy system as a network, which combines the extraction of primary fuels to the demand for energy services or energy consuming materials through a number of conversion and transport technologies. The model covers the development of a national or large regional energy system over some decades, i.e. the lifetime of large energy conversion equipment, or the time horizon of the market penetration of new energy consuming appliances in households or industries. The model was developed as the supply part of the energy model complex of the Commission of the European Communities and has been used for a number of studies since the 1970s.

The model was used for the assessment of national and Community emission reduction strategies under the JOULE Energy Research Programme of the European Commission. The present EFOM-ENV version was developed from a study of CO_2 reduction strategies for the European Commission. In the ongoing JOULE Framework Programme, further development of this tool is included in a Tool Package of the pilot phase of the EURIO Project, which is a part of an initiative for a European Energy-Environment-Economic Forum that has the objective to be a support for Research and Technology Development (RTD) in the fields of energy planning and environmental protection.

The database contains the network description of the energy system. Extraction, conversion and demand technologies are described by parameters concerning capacity, availability, thermal efficiency, flow or capacity costs, emissions of pollutants, and demand variations over time.

The model is demand driven. Scenario data concerning price and demand and forecast are either a part of the database or included in the scenario specification that is used for the LP matrix generator. LINPROG is one of the options for the LP software.

The EFOM software was developed in the 1970s for large mainframes, when the media for data communication were punchcard input and lineprinter output. The FORTRAN code is now available in a portable version, which has been compiled by various FORTRAN systems for mainframes, workstations and PCs. In addition to the traditional FORTRAN based software, EFOM was also developed in general model languages, most recently in GAMS [34].

The primary model interfaces for all these systems are ASCII files in specified formats. ASCII files in an appropriate format remains the best choice for the model interface, because they can be treated by any software.

However, the appropriate model interface is not necessarily a good user interface. To handle a large amount of data or many scenarios of the same application model written in the EFOM software, it is necessary to automate this data communication. A wide range of spreadsheet and database software have been used for this purpose by the users of the EFOM model.

7.2 Spreadsheet user interface

A spreadsheet user interface to handle assumptions and results of many scenarios for a model written in the EFOM software was developed over some years as a byproduct of the multinational studies and additional national applications of EFOM. Because different users are using different spreadsheets, it is necessary that such spreadsheet user interfaces are portable among the different releases of existing and future spreadsheet software.

A set of macro libraries was developed in Lotus 1-2-3 Release 2.x as a tool for transferring data between various data sources and the EFOM model. The main features are

- Data describing model assumptions are transferred from source tables into a standard database format
- Input formats required by the model are created from the data in a standard database format
- Output from the EFOM report writer is imported to a spreadsheet in database or table format that enable comparison of different scenarios
- Presentation tables and graphs are created for analysis and comparison of scenarios
- Documentation of assumptions and results

Examples of these graphical presentations are shown in Figure 10 and Figure 11 in Section 8.4.3. The macro library was developed in a low version of Lotus 1-2-3 for DOS. The use of new features in later releases were avoided in order to maintain portability to the wide range of new releases and competing spreadsheet software, e.g. Microsoft Excel or Borland QuattroPro.

The macro language of spreadsheet software, however, can be very sensitive to compatibility problems with newer releases and competing spreadsheet products. Instead of macros, formulas may be used to change the format of data between the ASCII files that are used for model interface and the input sheets or presentation tables or graphics that are the most important types of user interface. Tables linked by formulas developed in a low version of Lotus 1-2-3 will without difficulties be read and modified by all new releases of the major spreadsheet systems. Macros

or manual operations will still be needed to communicate with the model software using ASCII files.

7.3 Computer resources and organizational requirements

The traditional application of the EFOM model covers most of the subsystems for a single country and four to six 5-years periods (cf. Figure 6 in Section 8.2.1). The LP problem for this system contains between 5,000 and 10,000 rows and a similar number of columns. Solving this problem in the late 1980s would require 1–2 CPU-hours at a larger VAX system or 10–15 minutes at a large university mainframe. Today, a 486 PC will require about the same time. However, to maintain and manage a model of this volume of information will continue to require a large organization and many man-years.

For many types of studies much smaller models that cover only a single sector of the energy system will be far more useful. In the model for the electricity system, which is described in Section 8.4, the size of the LP matrix is less than 2,000 rows, which is solved at a good standard PC in less than 2 minutes and much faster at a workstation. This will allow running multiple scenarios for testing system boundaries and assumptions, parameter studies, and the analysis of the robustness of the results. With access to an existing database of techno-economic and structural parameters, it is possible for individuals to use this type of models for a wide range of purposes.

7.4 Dissemination of model software

LINPROG is available from Risø National Laboratory as a software package. It is distributed either as a stand alone tool, or it is included in a software package containing the EFOM model as a source code that is portable to several computer systems.

Since 1992 the price of a permanent license for LINPROG has been 500 ECU (about US $ 650) for a single user license.

We are practising the licensing policy for multi-user and site licenses: There is a basis price covering a single-user license, two licenses of the same system version allow installation at a multi-user environment (a PC network or a UNIX workstation), and three or more licenses will be a site license for LINPROG in one or more system versions.

The basis price is the one that is used for LINPROG. This price will apply for a single-user license for a software of an appropriate size, e.g. the LINPROG LP solver, or "EFOM Portable Source Code".

The LINPROG Linear Programming solver is developed and completely owned by Risø. The software for the EFOM model was developed as a part of the Energy Research Programmes of the European Commission. In collaboration with the Commission (DG XII/F/1), Risø has modified the code into a version that is portable to most FORTRAN system. This version is available on a single diskette, labelled "EFOM Portable Source Code", which contains the source code, a compiled version by e.g. Lahey F77L-EM/32 and command files for running the model and for compilation by different systems, e.g. VAX/VMS. The copyright of Risø for the "EFOM Portable Source Code" is thus limited to the structure of the diskette, some auxiliary code and the compilation, but not the contents of the source code itself.

Demonstration copies of both packages may be available on agreement for potential users as "invited shareware". "Shareware" means users are invited to test

the software, but those who want to use it regularly, need to pay a license fee. "Invited shareware" is our suggestion for the distribution form within a limited group of users in order to support the dissemination of modelling tools and encouraging further development.

8 Applications of LINPROG

In this chapter we describe the use of the model environment of LINPROG and some applications for energy system modelling. The early applications of LINPROG that were developed at Risø include a nuclear fuel shuffling and inventory model, and a one-period energy system model combining an input-output model and a network of energy flows in the refinery sector. The more recent examples describe various uses of the EFOM model software.

8.1 Early applications of LINPROG

8.1.1 Nuclear fuel shuffling and inventory model

One of the earliest LP applications at Risø was made by S. Weber (unpublished internal report, 1982) in his SOFIE model for planning optimal schemes for fuel management in nuclear reactors (SOFIE = SOphisticated Fuel Inventory Evaluation). The original idea in the SOFIE model dates back to Sauar [35], who proposed to minimize the total present value of the fuel cycle cost for a number of cycles. The loading, shuffling and discharge of fuel elements were decided for a few-region radial core model.

Linearization of this model resulted in an LP problem, which was solved by a predecessor version of LINPROG. The output from the model was a fuel loading scheme for regions and cycles, the burn up of the fuel elements, and the kWh price for each cycle and the whole period. The cash flow for each year was given for all phases in the fuel cycle. In some studies indivisibility constraints called for a BMILP (Binary Mixed Integer LP) extension of the optimization problem.

8.1.2 Energy rationing model

In 1986 Munksgaard Pedersen [32] used LP as a tool to analyze the possibilities for an optimal allocation of scarce energy resources in the event of a short-term reduction in oil supply. The result of this study was a model — LINRAT — which can be used as a decision aid for the optimal allocation of scarce energy resources according to a set of criteria defined by the user.

LINRAT is an input-output model for a single period within the tradition of economic modelling. As the oil disruption is assumed to last a short period, investment cannot take place, and, therefore, the technical coefficients in the model can be assumed to be constant. The official Danish input-output tables and energy matrices were used as the main data source for the model [36].

The objective function for the LP model is flexible and may consist of linear combinations of criteria. In the reported study, the aggregated employment was the main element to be maximized. The constraints are organized into four modules:

a The input-output module is a disaggregated account system in monetary units for 14 non-energy flows and the total energy flow. Import of non-energy is treated endogenously and import of energy exogenously. Final demand is split into: private consumption, government consumption, investment, export, and change in stocks.

b The substitution module splits the total energy balance into balances for 10 types of energy. Substitution among the various types of energy and energy savings in the production system is allowed.

c The refinery module is a network description of energy flows in energy units from crude oil to oil products. It handles two qualities of crude oil. Cracking

of crude oil represents a flexibility, so that the mix of oil products is not fully determined by the quality of the crude oil.

d The rationing module contains constraints on the consumption of the various energy products in the non-energy sectors.

LINRAT includes one scenario for the undisturbed economy and another one for a crisis economy.

The combination of an input-output model for most sectors of the economy and a network representation for sectors consisting of identifiable technologies may also be used for other purposes, e.g. emissions projections. For a wide range of studies, it will be more interesting to use linear programming to study the conditions for a feasible solution of the problem, rather than maximizing an arbitrarily specified objective function.

8.2 A network description of energy systems

Optimization models for national energy systems has been used widely during the 1980s for national energy planning purposes and for international comparative studies by international organizations. Typical study objectives have been the penetration of new technologies, impact of fuel price changes and optimal emissions reduction strategies. These models are also used for national, regional, or local energy planning studies, and studies of the technology choice in large energy consuming industries, e.g. cement, steel, pulp and paper, etc.

8.2.1 Application of the EFOM software

The EFOM model was used during the 1980s for reference projections of the energy systems in the member countries. Scenario assumptions concerning economic growth levels, oil import prices, and the role of solid fuels and nuclear power were studied. An extension to include emissions of pollutants and abatement techniques was implemented for all member countries and some countries outside the Community for studies on emissions reduction strategies.

The overall structure of the EFOM-ENV model is shown in Figure 6. The energy system is described as a network combining the extraction of primary fuels through a number of conversion and transport technologies to the demand for energy services or large energy consuming materials. Some of the subsystems also contain emission abatement technologies for SO_2 or NO_x. Each of the subsystems may consist of a large number of links, which refer to a database that contains the network information (upstream and downstream nodes) and a number of parameter values concerning capacity, availability, thermal efficiency, flow or capacity costs, emissions of pollutants, or daily and seasonal variations over time. Links representing primary fuels contain fuel price forecasts. The system is demand driven by forecasts of the demand for useful energy, energy services, or large energy consuming materials.

Being an energy flow model EFOM is characterized by many topological network constraints and balance equations. It is not a pure network model , however, and this excludes a direct optimization on network by specialized software. (The network integration technique by Tomlin and Welch [37] does not apply to EFOM either.)

As the LP formulation of an EFOM problem follows the energy system representation closely, there will be many redundant equations and unknowns in the initial formulation. These may be eliminated by the presolve module of LINPROG, see Section 4.7.

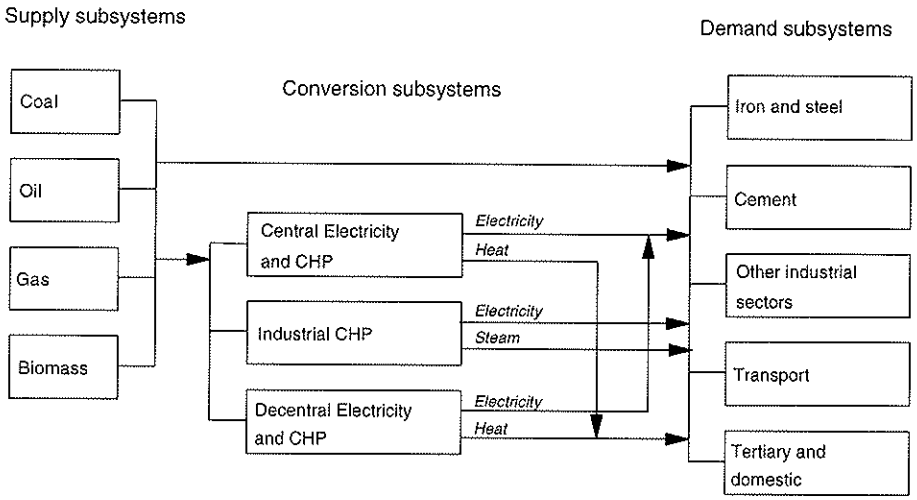


Figure 6. Modular structure of the EFOM-ENV model

8.2.2 The power generating system with CHP

The power generating system is the central part of the energy system that is described here, including combined heat and power (CHP) either for industrial steam raising or district heating.

Electricity and heat load variations are modelled using two-step load duration curves for the winter and summer seasons, leading to four annual flows for these energy carriers [38].

Figure 7 shows the generating technologies in the Central Electricity and CHP Subsystem. There are three types of electricity generating technologies that may serve the national or international transmission network with base load. These stations may also serve large urban district heating grids. Each of these types of power stations may be fed by coal, fuel oil or natural gas. There is a maximum share of coal, because a few per cent oil or gas is needed for start-up. In addition to the multi-fueled base-load units, there are oil or gas fired peak-load units and small-scale renewables connected directly to the regional transmission network.

8.2.3 Scenarios optimization by linear programming

All this information is converted into a standard LP constraint matrix, which is then processed by a general linear programming solver.

Scenarios for a period of 20-40 years divided into 5-year periods are defined by demand forecasts, energy prices, infrastructure constraints, emission regulations, capacity development plans, etc. The matrix generator creates constraints for each period, and the optimum for each scenario is found by minimizing the discounted costs for the total system over the period.

8.2.4 Limitations of optimization models

This model belongs to the "bottom-up" approach of techno-economic models, which is best suited for describing the potentials and least-cost mixes of identifiable technologies producing one or few commodities, and covering a reasonable part of

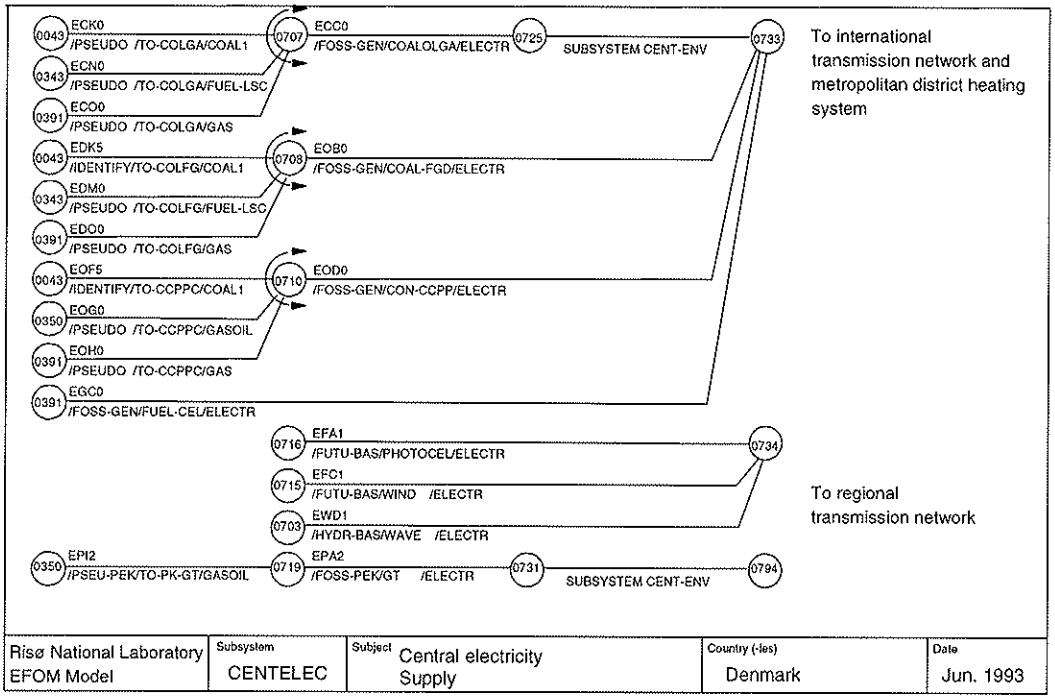


Figure 7. Detailed network chart for the central electricity subsystem, supply side

the energy system. Both large-scale technologies, e.g. power plants, and small-scale technologies that are used in large quantities, e.g. individual heating systems or electric appliances, are such identifiable technologies. In contrast, the description of energy use and emissions caused by production and consumption of the variety of goods and services in a modern economy requires a "top-down" approach from the econometric or macroeconomic tradition based on estimation of long time series to disclose the consequences of technology development and behaviour of economic agents [39].

8.3 Cost curves for emissions reduction

An example of an application of the EFOM model is the construction of cost curves for the reduction of emissions of major atmospheric pollutants. Cost curves show the increasing value of the objective function for a series of scenarios for the same description of the energy system, where the constraint on one or more pollutants becomes more tight.

The EC cost-effectiveness analysis of CO_2 reduction options was a part of study under the JOULE Energy Research Programme of the Commission of the European Communities concerning options for CO_2 emissions reduction [40]. The method of cost curves was developed for an earlier study on abatement strategies for SO_2 and NO_x emissions [41]. The cost curve approach has since been used for several international collaborative studies, e.g. The UNEP Greenhouse Gas Costing Study [42], which has developed a set of guidelines for the construction of cost curves.

8.3.1 Discounted energy system costs

The central definition of cost-effectiveness was the discounted cost of the energy system within the limits set by the model description of the system for any feasible reduction level of a particular emission item within the period. A key result that complies to this definition is shown in Figure 8. The value of the objective function

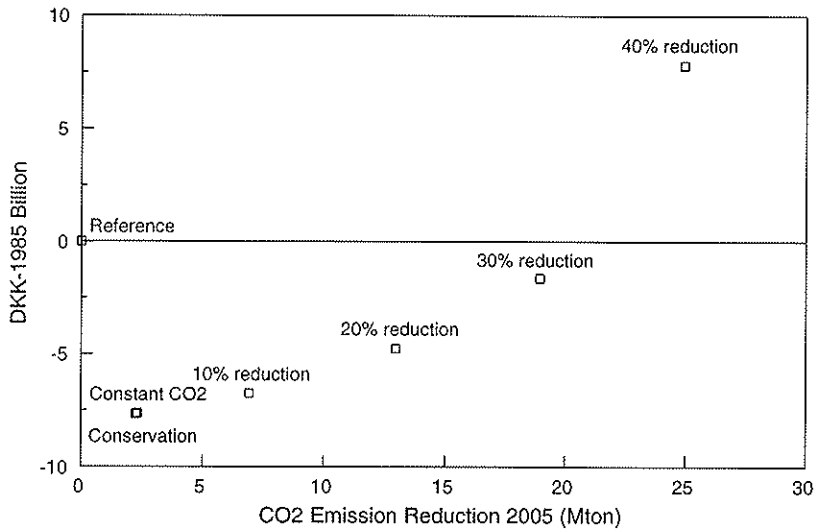


Figure 8. CO_2 emissions reduction and increase in discounted energy system costs for Denmark by 2005, 5% discount rate

of the LP problem (i.e. the discounted energy system costs for the period 1985–2010) is shown for selected values of CO_2 reductions by 2005 compared to the 1988 level.

8.3.2 Scenario assumptions

The Reference Case contains the penetration of expected new technologies and the emissions abatement measures that is required by the existing legislature, and some energy conservation measures that are assumed in the demand forecasts.

The Conservation Case is the reference case with penetration of further conservation measure that are options for the optimization, e.g. more efficient domestic electric appliances. It represents a "least cost planning" approach, assuming that conservation measures are subject to the same kind of rationality as utility planning.

The latter scenario is the starting point for introducing the emission constraints, "Constant CO_2" through "40% reduction", in the optimization. Further reduction constraint would lead to an infeasible solution of the LP problem.

8.3.3 Assumptions for the national study

The assumptions concerning the development of demand for energy services and fuel import prices are those of the harmonized multinational study.

Techno-economic data for energy technologies, including the technical options for emission reduction, were based on the earlier studies or detailed studies within the present multinational study.

Pure national data describing the structure of the energy system and detailed assumptions for the demand for energy services were specified to comply to national statistics and the assumptions used for national energy planning. Further political constraints were avoided in order get the most true optimization results.

In addition to the constraints that express important energy policy issues, a number of constraints are necessary to prevent unrealistic model results. Most of these quantify infrastructural constraints or long-term contracts.

8.3.4 Overall conclusion and policy recommendations

The main conclusion of the study for Denmark was that it would be possible to optimize the Danish energy system over a period to achieve a target for a substantial reduction of national CO_2 emissions, at extra costs no larger that could be outweighed by some cost-effective energy savings measures. Such a conclusion is controversial; in particular the jump in cost from the Reference to the Conservation scenario.

The main elements in the CO_2 reduction policy should be energy conservation, intensive use of natural gas for electricity generation, and penetration of renewables. Further CO_2 reductions would require that also the use of natural gas be reduced and, thus, replaced by non-fossil fuel technologies. The results for Denmark were similar to those of most of the other EC countries.

However, using these results as a general policy recommendation may be counterproductive for a target of cost-effective CO_2 emissions reductions. In this study the optimum would lead to a massive substitution of gas for coal in the power sector. This is actually happening, in particular in countries where the electricity supply industries have been liberalized. Sooner or later this substitution may lead to natural gas prices higher than those assumed for the optimization.

It should be emphasized that the outcome of any model study is the understanding of the behaviour of the system under specified conditions rather than recommendations to be implemented according to an optimal solution.

8.3.5 Interpretation of dual values

The LP concept of dual values is defined as the addition to the value of the objective function per unit of a given constraint (cf. Section 3.4). This concept was not used for the study just referred.

Figure 9, however, shows an attempt for an interpretation of the dual values

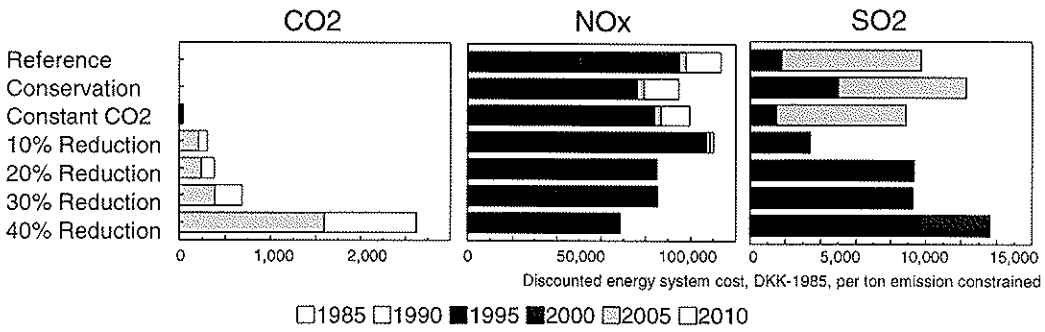


Figure 9. Increase in discounted energy system costs for Denmark per unit of emissions constrained, 5% discount rate

of the constraints for CO_2, NO_x, and SO_2 emissions. All emissions are unconstrained in 1985 and 1990. The CO_2 emission constraints describe the scenarios. The CO_2 emissions are unconstrained in the Reference and Conservation scenarios. In all other scenarios the CO_2 emissions are constrained from 1995 onwards at the 1988-level or reduced by a percentage from 2005. NO_x and SO_2 emissions are constrained from 1995 onwards for all scenarios at decreasing values according to the EC Directive for Large Combustion Installations from 1988.

The constant CO_2 constraint is effective by 1995 and 2000, but adds very little to the objective function. The reduction constraints are effective both in 2005 and 2010. The increasing costs reflect the shape of Figure 8: Increasing costs as the constraint becomes more tight. The 2005 constraint will add more to the objective function that the 2010 constraint, because investment in new equipment will be required earlier.

The NO_x emission constraints are effective in all years when the CO_2 constraint is not very tight. At the more tight CO_2 constraints late in the period, the NO_x emission is reduced below the limit. The very tight CO_2 emissions late in the period will make the modest NO_x emissions constraints by 1995 more costly, because this constraint will require investment in equipment that may not be necessary later. The same mechanism is found for the SO_2 emissions.

8.4 The international market for electricity

The following example shows an application of a model developed as a sectoral module of the model described in the previous section for analysing a regional or national electricity system over 20–30 years, emphasizing the influence of changing market conditions for the choice of new generation capacity and fuels.

8.4.1 A sectoral model for electricity

This model is organized as described in Table 10 in Section 7. There is a general database containing the energy flow network and the techno-economic values, which may be used for different countries or system boundaries. The model is run using national or regional specific data for initial capacities and infrastructural constraints, and a set of scenario assumptions, i.e. demand forecasts, energy prices, emissions constraints, etc.

The user interface for each set of data is a spreadsheet model that also contains various tools for documentation of scenario assumptions and graphic presentation of assumptions and results.

The model is developed from the study of CO_2 reduction strategies described in the previous sections, and most of the assumptions for the reference case are the same as those described for that study. This model is a sectoral model for the electricity sector including CHP and district heating.

8.4.2 Danish utilities in a competitive market

Traditional energy planning in most countries has been based on the idea of national autarky. Although a lack of indigenous resources of fossil fuels makes fuel import necessary in many countries, the electricity demand is generated nationally, most often by nationalized or public owned utilities. However, electricity import to Denmark has been considerable in some years, whenever hydro power has been abundant in Norway and Sweden.

The recent development towards competition on a national and European scale for the electricity supply industry is studied in a project for the Danish Energy Research Programme with participation from Local Governments' Research Insti-

tute (AKF), Roskilde University and Risø. The main task of Risø was to adapt our modelling experience to a quantitative analysis of a new organizational framework.

There is no single tool available for this analysis. The existing well-established methods of analysis will give only partial answers, and the methods of analysis of electricity markets and the behaviour of producers and consumers are only in their infancy.

To analyse the long-term consequences three different analytical or modelling approaches with increased analytical sophistication are being used:

- short-term and long-term cost of competing technologies
- an engineering model with an optimization of a network of energy flows, and
- an econometric, partial equilibrium model for the electricity market

The first two approaches follow the logic of the traditional organization of the industry, where vertically integrated monopolies with reserve capacity traded according to some agreed principle. The econometric models are designed for the logic of the marketplace to analyse the behaviour of many economic agents in order to find a clearing price through the interactions of supply and demand. There is a long tradition for these models, which is also supported by international cooperation on model development and elaborate international standards for national accounts statistics. The available econometric or macroeconomic models, however, do not offer a sufficient representation of the options for technology choices among the most important technologies within the energy sector.

For this purpose the engineering modelling approach is needed, in which technological options are physically identified. The EFOM model is used as a tool for this analytical approach to study the consequences of changes in parameters that describe the different market frameworks.

8.4.3 Sensitivity to economic parameters

Figure 10 shows some results of an application of this modelling approach for analysing the Danish electricity system until 2010, emphasizing the impact of changing market conditions on the choice of new generating capacity and fuels. Most of the assumptions for the reference case are the same as those of the multinational study of CO_2 reduction strategies described in the previous section.

The parameter variations concern:

- limits for electricity import and export
- import and export prices
- discount rate

In the Reference Scenario the Danish electricity system is optimized for the period 1995–2010 assuming only contracted import and export. Import and export prices are set constant over the year. The import price was chosen at 0.15 DKK/kWh for the whole period, and the export price should reflect long-term marginal costs for a reference technology (0.28–0.32 DKK/kWh). The objective function is the total discounted costs for the period at 5% discount rate. A set of constraints reflect the infrastructure of the Danish electricity and CHP system. International electricity trade is constrained at a minimum.

In the next scenario "Import" maximum import is set at the capacity of the transmission lines from 1994. The third scenario "More trade" assumes further expansion of the transmission capacity. In the scenario "Mid-price" the same price is set for both import and export. The last scenario is identical to "Mid-price",

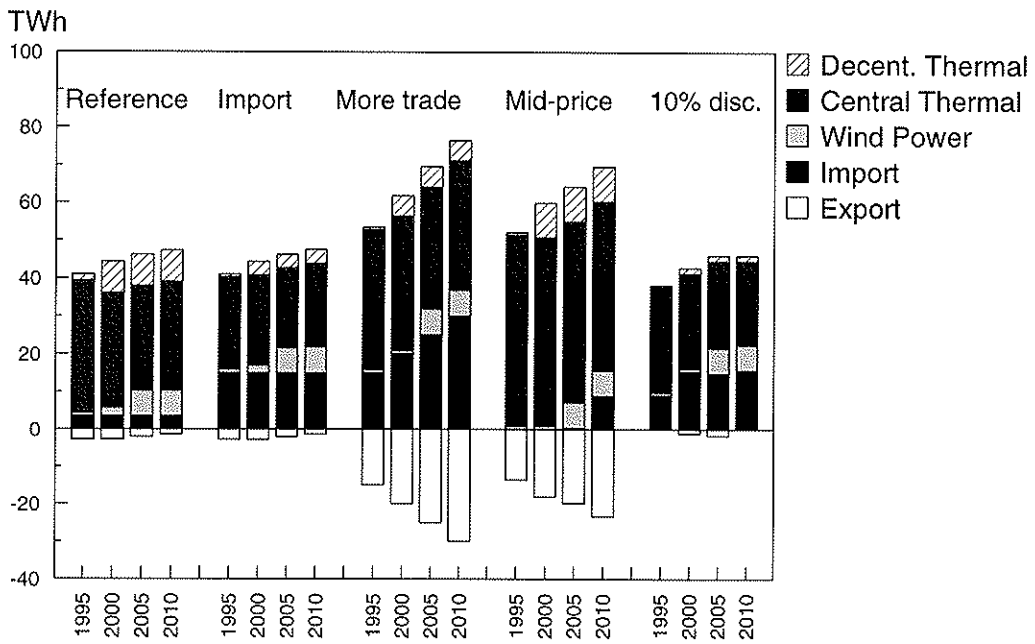


Figure 10. Electricity generation technologies in Denmark 1995–2010

but the discount rate is set at 10% in order to reflect the financial conditions for utility investment at a competitive market for electricity.

The optima for all the scenarios with unconstrained trade volumes are very sensitive to changes of the import and export price assumptions. It shows that most electricity is generated at central thermal stations, either as combined heat and power or electricity-only generation. The volume of the latter will mirror the variations in import and export volumes.

Figure 11 shows the annual expenditure for fuel, operation and maintenance

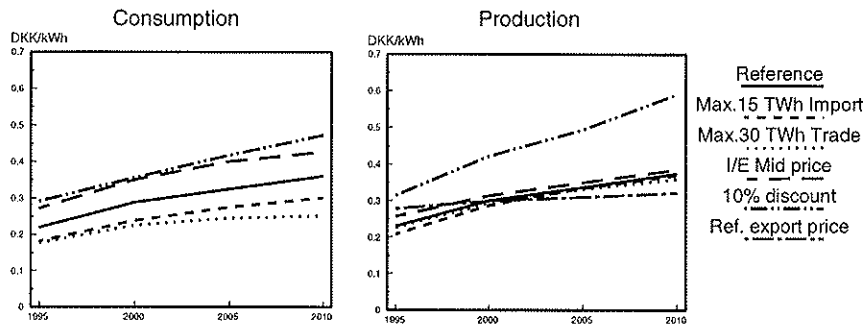


Figure 11. Average expenditure for electricity consumption and electricity production in Denamrk 1995–2010 at optimal development

and investment in new capacity per kWh in each year for the optimal development under the various scenario assumptions. Obviously, these expenditure will normally be increasing, when new capacity is required for an optimal long-term

development of the industry, because no expenditure to meet existing financial obligations from existing equipment are included.

The cost concept shown here is undiscounted cost per unit for each period. They are calculated from optimal scenarios in which the discounted costs over all periods were minimized.

References

[1] P. Kirkegaard and O. L. Rasmussen, "LINPROG: A linear-programming code developed at Risø," Tech. Rep. Risø-M-2797, Risø National Laboratory, DK-4000 Roskilde, Denmark, 1990.

[2] J. J. Dongarra and E. Grosse, "Distribution of mathematical software via electronic mail," *Comm. ACM*, vol. 30, pp. 403–407, 1987.

[3] R. G. Bland, "The allocation of resources by linear programming," *Scientific American*, vol. 244, pp. 108–119, Jun 1970.

[4] J. L. Nazareth, *Computer Solution of Linear Programs.* New York and Oxford: Oxford University Press, 1987.

[5] *IBM Mathematical Programming System Extended/370 (MPSX/370), Primer.*, 1979. GH19-1091-1, File No. S370-82.

[6] B. A. Murtagh and M. A. Saunders, "MINOS 5.1 USER'S GUIDE," Tech. Rep. SOL 83-20R, Systems Optimization Laboratory, Stanford University, California 94305-4022, 1987.

[7] G. B. Dantzig, *Linear Programming and Extensions.* New Jersey: Princeton University Press, 1963.

[8] B. A. Murtagh, *Advanced Linear Programming: Computation and Practice.* New York: McGraw-Hill, 1981.

[9] G. H. Golub and C. F. van Loan, *Matrix Computations.* Baltimore, Maryland: The John Hopkins University Press, 1984.

[10] G. E. Forsythe and C. B. Moler, *Computer Solution of Linear Algebraic Systems.* Englewood Cliffs, N.J.: Prentice-Hall, 1967.

[11] W. Orchard-Hays, *Advanced Linear-Programming Computing Techniques.* New York: McGraw-Hill, 1968.

[12] M. Benichou, J. M. Gauthier, G. Hentges, and G. Ribiere, "The efficient solution of large-scale linear programming problems — some algorithmic techniques and computational results," *Mathematical Programming*, vol. 13, pp. 280–322, 1977.

[13] I. S. Duff and J. K. Reid, "An implementation of Tarjan's algorithm for the block triangularization of a matrix," *ACM Transactions on Mathematical Software*, vol. 4, pp. 137–147, 1978.

[14] I. S. Duff, "On algorithms for obtaining a maximum transversal," *ACM Transactions on Mathematical Software*, vol. 7, pp. 315–330, 1981.

[15] E. Hellerman and D. C. Rarick, "The partitioned preassigned pivot procedure (P^4)," in *Sparse matrices and their applications, Proceedings of Conference in Yorktown Heights, September 9-10, 1971* (D. J. Rose and R. A. Willoughby, eds.), pp. 67–76, Plenum Press, 1972.

[16] J. A. Tomlin, "Modifying triangular factors of the basis in the simplex method," in *Sparse matrices and their applications, Proceedings of Conference in Yorktown Heights, September 9-10, 1971* (D. J. Rose and R. A. Willoughby, eds.), pp. 77–85, Plenum Press, 1972.

[17] J. J. H. Forrest and J. A. Tomlin, "Updated triangular factors of the basis to maintain sparsity in the product form simplex method," *Mathematical Programming*, vol. 2, pp. 263–278, 1972.

[18] R. H. Bartels, J. Stoer, and C. Zenger, "A Realization of the Simplex Method based on Triangular Decompositions," in *Handbook for Automatic Computation - Linear Algebra* (J. H. Wilkinson and C. Reinsch, eds.), pp. 152–190, Springer-Verlag, 1971.

[19] P. M. J. Harris, "Pivot selection methods of the Devex LP code," *Mathematical Programming*, vol. 5, pp. 1–28, 1973.

[20] I. Maros, "A general phase-I method in linear programming," *European Journal of Operations Research*, vol. 23, pp. 64–77, 1986.

[21] D. Goldfarb and J. K. Reid, "A practicable steepest-edge simplex algorithm," *Mathematical Programming*, vol. 12, pp. 361–371, 1977.

[22] R. E. Bixby, J. W. Gregory, I. J. Lustig, R. E. Marsden, and D. F. Shanno, "Very large-scale linear programming: a case study in combining interior pointy and simplex methods," *Operations Research*, vol. 40, pp. 885–897, 1992.

[23] R. S. Garfinkel and G. L. Nemhauser, *Integer Programming*. New York: John Wiley and Sons, 1972.

[24] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, "A practical anti-cycling procedure for linear and nonlinear programming," Tech. Rep. SOL 88-4, Systems Optimization Laboratory, Stanford University, California 94305-4022, 1988.

[25] J. A. Tomlin, "On scaling linear programming problems," *Mathematical Programming Study*, vol. 4, pp. 146–166, 1975.

[26] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*. Oxford: Clarendon Press, 1986.

[27] M. H. van Emden, "Increasing the efficiency of quicksort," *Comm. ACM*, vol. 13, pp. 563–567, 1970.

[28] J. A. Tomlin and J. S. Welch, "Formal optimization of some reduced linear programming problems," *Mathematical Programming*, vol. 27, pp. 232–240, 1983.

[29] E. van der Voort, E. Donni, C. Thonet, E. B. d'Enghien, C. Dechamps, and J. F. Guilmot, *ENERGY SUPPLY Modelling Package EFOM-12 C Mark I - Mathematical description*. Louvain-la-Neuve, Belgium: CABAY (for the Commision of the European Communities), 1984.

[30] C.-E. Fröberg, *Lärobok i numerisk analys* (In Sweedish). Copenhagen and Stockholm: Scandinavian University Books, Svenska Bokförlaget/Bonniers, 1962.

[31] C. F. Cohn, *Algebra, Vol. 1*. Bedford: Bedford College, 1981.

[32] J. M. Pedersen, "LINRAT - en energirationeringsmodel for Danmark," Tech. Rep. Risø-M-2611, Risø National Laboratory, DK-4000 Roskilde, Denmark, 1986. (in Danish).

[33] D. M. Gay, "Electronic mail distribution of linear programming test problems," *Math. Program. Soc. Comm. Algorithms Newsl.*, vol. 13, pp. 10–12, 1985.

[34] H. de Kruijk, "The EU Energy and Environment Model EFOM-ENV Specified in GAMS: Model description and user's guide," Tech. Rep. ECN-C-94-021, Netherlands Energy Research Foundation, ECN, Petten, The Netherlands, 1994.

[35] T. O. Sauar, "Application of linear programming to in-core fuel management optimization in light water reactors," *Nuclear Science and Engineering*, vol. 46, pp. 274–283, 1971.

[36] Danmarks Statistik, København, *Nationalregnskabsstatistik 1966-81.* (in Danish).

[37] J. A. Tomlin and J. S. Welch, "Integration of a primal simplex network algorithm with a large-scale mathematical programming system," *ACM Trans. Math. Software*, vol. 11, pp. 1–11, 1985.

[38] P. E. Grohnheit, "Modelling CHP within a national power system," *Energy Policy*, vol. 21, no. 4, pp. 418–429, 1993.

[39] P. E. Grohnheit, "Economic interpretation of the EFOM model," *Energy Economics*, vol. 13, no. 2, pp. 143–152, 1991.

[40] *COHERANCE s.c., Cost-Effectiveness Analysis of CO_2 Reduction Options*, May 1991. Synthesis Report/Country Reports (Report for the Commission of the European Communities, DG XII, JOULE Programme. Models for Energy & Environment).

[41] *Energy and Environment - Strategies for Acid Pollution Reduction.* Brussels, 1989. Commission of the European Communities (ed.).

[42] *UNEP Greenhouse Gas Abatement Costing Studies Phase 2 Guidelines.* Risø National Laboratory, Denmark, May 1994.

A Report writer in FORTRAN

This is a print-out of a sample version of a report writer for LINPROG written in FORTRAN; explanations are found in the program comments:

```
      PROGRAM REPORT
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
C
C THIS PROGRAM READS THE LP SOLUTION FROM THE BINARY COMMUNICATION
C FILE PRODUCED BY LINPROG, AND PRODUCES A CORRESPONDING FORMATTED
C SOLUTION FILE. THIS FILE WILL CONTAIN KEY DATA FOR THE PROBLEM
C AND A TABULATION OF THE ROW AND COLUMN SECTIONS OF THE SOLUTION.
C MOREOVER, THE PROGRAM STORES ROW AND COLUMN NAMES AND ACTIVITIES,
C DUAL ACTIVITIES AND REDUCED COSTS, IN ARRAYS FOR LATER REFERENCE,
C IF YOU WANT A PARAGON FOR A FULL-FLEDGED REPORT WRITER
C
      PARAMETER (MR=16676, MC=15695)
      DIMENSION ACTIVR(MR), DUALAC(MR), ACTIVC(MC), REDUCO(MC)
C THESE FOUR ARRAYS WILL HOLD THE FOLLOWING INFORMATION:
C     ACTIVR(I)  =  ACTIVITY OF ROW NO. I
C     DUALAC(I)  =  DUAL ACTIVITY OF ROW NO. I
C     ACTIVC(J)  =  ACTIVITY OF COLUMN NO. J
C     REDUCO(J)  =  REDUCED COST FOR COLUMN NO. J
C
      CHARACTER ROWNAM(MR)*8, COLNAM(MC)*8
C ROWNAM(I) WILL CONTAIN THE 8-CHARACTER-NAME OF ROW NO. I
C COLNAM(J) WILL CONTAIN THE 8-CHARACTER-NAME OF COLUMN NO. J
      CHARACTER YYMMDD*6, VERS*4, TARGET*8, PBSTAT, MARK, STATUS*10
     1, SETNAM*8, OBJNAM*8, RHSNAM*8, RNGNAM*8, BDSNAM*8, STATUR*2
     2, STATUC*2
C OPEN FILES
      OPEN (UNIT=1, FILE='binout.dat', STATUS='OLD', ACCESS='SEQUENTIAL'
     1, FORM='UNFORMATTED')
      REWIND 1
      OPEN (UNIT=2, FILE='report.out', STATUS='UNKNOWN'
     1, ACCESS='SEQUENTIAL', FORM='FORMATTED')
      REWIND 2
C READ AND WRITE IDENTIFICATION SECTION
  100 READ (1,END=900) YYMMDD, VERS, TARGET
      WRITE (2,101) YYMMDD, VERS, TARGET
  101 FORMAT (//' PROBLEM DATE', T32, A/' LINPROG VERSION', T32, A/
     1' TARGET', T32, A/)
      READ (1) SETNAM, OBJNAM, RHSNAM, RNGNAM, BDSNAM
      WRITE (2,102) SETNAM, OBJNAM, RHSNAM, RNGNAM, BDSNAM
  102 FORMAT (' NAME OF DATA SET', T32, A/' NAME OF OBJECTIVE ROW', T32
     1, A/' NAME OF RIGHT-HAND SIDE', T32, A/' NAME OF RANGES', T32, A/
     2, ' NAME OF BOUNDS', T32, A/)
      READ (1) PBSTAT, L, NS, ICNTR, FUNVAL
      STATUS = 'INFEASIBLE'
      IF (PBSTAT .EQ. 'O') STATUS = 'OPTIMAL'
      WRITE (2,103) STATUS, L, NS, ICNTR, FUNVAL
  103 FORMAT (' PROBLEM STATUS', T32, A/' NUMBER OF ROWS', T32, I4/
     1' NUMBER OF COLUMNS', T32, I4/' NUMBER OF ITERATIONS', T32, I4/
     2' OBJECTIVE VALUE', T32, 1PE20.13)
C READ AND WRITE ROW SECTION OF SOLUTION
      WRITE (2,104)
  104 FORMAT (/' TABULATION OF THE FILED ROW SECTION'/T3, 'NAME', T11
     1, 'NUMBER', T18, 'STATUS', T27, 'ACTIVITY', T43, 'SLACK', T54
     2, 'DUAL ACTIVITY', T70, 'MARK')
      DO 1, I = 1, L
      READ (1) MARK, ROWNAM(I), STATUR, ACTIVR(I), SLACAC, DUALAC(I)
    1 WRITE (2,105) ROWNAM(I), I, STATUR, ACTIVR(I), SLACAC, DUALAC(I)
     1, MARK
  105 FORMAT (T2, A8, T12, I4, T20, A2, T25, 1PE13.6, T39, 1PE13.6, T54
     1, 1PE13.6, T71, A1)
C READ AND WRITE COLUMN SECTION OF THE SOLUTION
      WRITE (2,106)
  106 FORMAT (/' TABULATION OF THE FILED COLUMN SECTION'/T3, 'NAME', T11
     1, 'NUMBER', T18, 'STATUS', T27, 'ACTIVITY', T41, 'INPUT COST', T55
     2, 'REDUCED COST', T70, 'MARK')
      DO 2, J = 1, NS
```

```
      READ (1) MARK, COLNAM(J), STATUC, ACTIVC(J), COST, REDUCO(J)
    2 WRITE (2,105) COLNAM(J), J+L, STATUC, ACTIVC(J), COST, REDUCO(J)
     1, MARK
C NOW YOU MAY PROCEED WITH YOUR REPORT WRITER, USING THE SIX ARRAYS
C ACTIVR(), DUALAC(), ACTIVC(), REDUCO(), ROWNAM(), AND COLNAM():
*           ..........................
*           ..........................
*           ..........................
C GO BACK AND READ MORE SOLUTION SETS, IF ANY
      GO TO 100
C CLOSE COMMUNICATION FILE
  900 CLOSE (1)
C
      END
```

B Organisation of LINPROG source

Here we present a few details on how the LINPROG source code is organised. We use INCLUDE files for the COMMON declarations in the individual program units, although this adds to the very few LINPROG violations of Standard FORTRAN 77. Also an included file LINPAR.CMN with problem size parameters set by FORTRAN's PARAMETER statements is used. These included files reduce the repetitions in the source code and make the maintenance of the code both safer and easier. In particular it is very easy to increase parameters related to problem size in LINPROG. Examples of such parameters are MR and MC, which are the maximum number of rows and columns, respectively, for the LP matrix. This INCLUDE design for organising the code was patterned after the EFOM software [29].

In the following we list, in tabular form, the FORTRAN subprograms making up LINPROG itself and the machine-dependent auxiliary subroutines, together with a short description of their tasks. Apart from the main program, the lists are in alphabetic order.

Name	Task
MAIN	LINPROG driver program.
BNDINP	Reads BOUNDS Section input.
BTRAN	Makes extended BTRAN with concurrent Forrest-Tomlin updating.
CAPS	Converts lower-case letters in a string to corresponding upper-case.
CHECK	Checks the solution for feasibility.
CHUZC	Selects entering variable in Phase 1 or 2.
CHUZR	Selects leaving variable in Phase 1 or 2.
COLINP	Reads COLUMNS Section input and generates input matrix lists.
COLLAT	Computes integer equivalents of the two halves of an 8-byte character word.
COLOUT	Produces column section output.
COLPRT	Prints column section output.
COMPIL	Compiles keywords in Command File for one problem.
CRASH	Selects an initial triangular basis.
DELETE	Deletes an entry from double linked list during presolve.
EFFLEN	Finds the effective length of a string ignoring trailing blanks.
ELIMIN	Eliminates variable from constraint matrix during presolve.
ERRPRT	Prints error messages.
FILSOL	Produces a binary solution file.
FIND	Searches for matching string of a text.
FMONIT	Monitors the infeasibilities.
FTRAN	Makes an FTRAN step.

INBILS	Builds certain linked lists to be used for inversion.
INVERT	Makes an LU re-inversion.
LOAD	Loads information from restart file.
LPMAST	Produces LP matrix statistics.
LPSOLV	Governs the solution of the LP.
LUDOT	Computes scalar product of sparse eta vectors for the LU inversion.
MANCOL	Performs memory management in column-ordered list during inversion.
MANROW	Performs memory management in row-ordered list during inversion.
MDATE	Computes the day number in the year from the date.
NEW	Adds a new element to double linked list during presolve.
NUMEDI	Formatting routine for numbers.
NUMVAR	Returns external variable number, given its internal number.
OBJECT	Computes the objective function.
PIVCOL	Scans pivot column candidate to produce a column eta vector and a pivot element.
PIVROW	Scans pivot row to produce a row eta vector during inversion.
PRICE	Computes reduced costs.
RECBET	Reconstructs updated right-hand side β.
REDLNK	Makes double linked lists for constraint matrix during presolve.
REDMOD	Modifies lists and arrays for reduced matrix during presolve.
REDSAV	Saves original lists after presolve for the recreation process.
REDSIG	Searches for redundant inequalities by sign test during presolve.
REDSRW	Eliminates singleton rows during presolve.
REDSTA	Writes statistics for presolve process.
REDTRE	Eliminates nodal balance equations during presolve.
RESTOR	Restores original lists to be used in recreation.
RHSINP	Reads RHS Section input.
RHSOFF	Modifies rhs for offset in variable during presolve.
RNGINP	Reads RANGES Section input.
ROWINP	Reads Matrix File heading and ROWS Section input.
ROWOUT	Produces row section output.
ROWPRT	Prints row section output.
SAVE	Saves information to dump file.
SIGNUM	Makes a structural picture of the LP matrix.
SISPA0	Tries to eliminate zero slacks (Phase 0).
SISPAR	Performs Phase 1 or 2 of sparse revised simplex.
TABLIS	Prepares column ordered list for constraint matrix.
TIMJOB	Returns time of day in format hh:mm.
TRIN1	Sorting routine for one integer vector.
TRIN2	Sorting routine for two integer vectors.
UPDATE	Makes a single step in the updating of ordered lists for inversion.

Finally, a BLOCK DATA subprogram provides default settings of the FORTRAN unit numbers for the program files.

LINPROG uses three auxiliary subroutines whose coding depend on the actual type of computer and the FORTRAN compiler:

DATJOB	Returns current date.
MCLOC	Timing routine.
OPENLP	Performs proper opening of LINPROG files.

C Installation of LINPROG

The installation of LINPROG depends on the type of computer in question. An executable code file ("linprog" or "linprog.exe") may just be copied to some suitable subdirectory in your computer system; if you follow the suggestions given in Chapter 5, this would be "code" beneath "lp".

A LINPROG source program must be compiled and linked using some locally based FORTRAN 77 compiler. We give below examples how this might be done on three different computer systems: DOS for PC, Unix, and DEC VAX/VMS. We will still use the directory structure in Chapter 5, supplemented by another directory "auxil" at the "lp" level, which contains the auxiliary machine-dependent routines DATJOB and MCLOC (Appendix B).

DOS example: We assume that Lahey's compiler F77L-EM32 with DOS extender is used. Then we could use the following .BAT file:

```
f7713 linprog /ns /4 > linprog.msg
f7713 licens /ns
f7713 openlp.std /ns
f7713 ..\..\auxil\datjob.lah /ns
f7713 ..\..\auxil\mcloc.lah /ns
copy ..\..\auxil\*.obj *.*
if errorlevel 1 goto quit
386link linprog licens openlp datjob mcloc -pack -nomap -stub runb,vmmb
cfig386 linprog -nosignon
:quit
```

UNIX example: Here the corresponding task is conveniently carried out by a makefile, where all the COMMON include files are collected in a macro:

```
# macro for common include files
cmn = ANONI1.CMN ANONI2.CMN COMARR.CMN COMINT.CMN COMNAM.CMN \
COMPR.CMN COMRES.CMN COMSIM.CMN KEYPAR.CMN LIDE.CMN LIDI.CMN \
LINPAR.CMN LIS.CMN LIV.CMN NUMNAM.CMN PERI.CMN PICTUR.CMN YEARDA.CMN

# macro for object modules
objects = linprog.o licens.o openlp.o datjob.o mcloc.o

linprog: $(objects)
<tab> f77 -o $@ $(objects)

linprog.o: linprog.f $(cmn)
<tab> f77 -O -c linprog.f
licens.o: licens.f
<tab> f77 -c licens.f
openlp.o: openlp.f PERI.CMN
<tab> f77 -c openlp.f
datjob.o: ../../auxil/datjob.f
<tab> f77 -c ../../auxil/datjob.f
mcloc.o: ../../auxil/mcloc.f
<tab> f77 -c ../../auxil/mcloc.f
```

VAX/VMS example: A DCL command file (with extension .COM) for compiling and linking LINPROG may look as follows:

```
$fortran/list linprog
$fortran licens
$fortran openlp.vax
$fortran [--.auxil]datjob.std
$fortran [--.auxil]mcloc.vax
$link linprog, licens, openlp, datjob, mcloc
$exit
```

Note that VAX FORTRAN provides two kinds of Double Precision: D-floating and G-floating. G-floating has slightly less precision but a much larger number range than D-floating. It is important that a report writer be compiled with the

same Double Precision type as was used for LINPROG. D-floating is the default type, and we have stuck to this choice when compiling LINPROG at Risø.

As mentioned in Appendix B the include file LINPAR.CMN contains a number of parameters for controlling the maximum problem size. These are typically array bounds which are set by the FORTRAN 77 statement PARAMETER, for example MR = the maximum number of LP rows. If some of these bounds must be increased, only this single file need to be modified prior to the recompilation. On the other hand, computer limitations may sometimes force a lowering of the bounds in LINPAR.CMN to let the resulting code fit into a specific computer system, or let it run faster.

D LP vocabulary

Activity:	Value of some LP variable.
Basic index set:	The set of variable indices corresponding to the current basis.
Basis:	The subset of the LP variables that determine the current solution are said to be *basic* or to form the *basis*. The precise definition is given in Section 4.1.
Constraint Matrix:	Totality of the coefficients a_{ij} in (1).
Feasibility:	A set of variables is said to define a *feasible point* or a *feasible solution* if all constraints are satisfied.
Infeasibility:	We have *infeasibility* when not all constraints can be satisfied.
Inversion:	The subset of basic variables defines the *basis matrix*. From this we may obtain the solution by a matrix inversion. This is discussed in Section 4.4.

Index

— E —

— F —

— G —

— H —

— I —

— K —

— L —

— M —

— N —

— O —

— P —

— Q —

— R —

Bibliographic Data Sheet **Risø–R–707(EN)**

Title and author(s)

LINPROG - A Linear Programming Solver
Mathematical description and model applications

Peter Kirkegaard and Poul Erik Grohnheit

ISBN	ISSN
87–550–1926–9	0106–2840

Dept. or group	Date
Computer Section and Energy Systems Group	June 1995

Groups own reg. number(s)	Project/contract No.

Pages	Tables	Illustrations	References
103	10	11	42

Abstract (Max. 2000 char.)

A FORTRAN 77 computer code LINPROG has been developed at Risø for solving medium- to large-scale linear programming problems. It runs on a wide range of computer platforms, including PC's. LINPROG uses the revised simplex method with the Forrest-Tomlin updating scheme of the inverse basis. Sparse-matrix techniques are applied throughout. An efficient presolve module is incorporated. A comprehensive test and verification study has been performed with data sets provided by local users and in the literature. The computer environment for different platforms are described as well as the design of models that is needed for generating large linear programming problems. Examples of model studies using LINPROG as a solver are given, focusing on solutions of optimization models for national energy systems that are described as a network of energy flows, where discounted costs over 20-30 years are minimized. Solution of optimization models for national energy systems and for energy planning is a main application field of LINPROG.

Descriptors INIS/EDB

Available on request from:
Information Service Department, Risø National Laboratory
P.O. Box 49, DK–4000 Roskilde, Denmark
Phone (+45) 46 77 46 77, ext. 4004/4005 · Telex 43 116 · Fax (+45) 46 75 56 27